Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tim Cestnik

LAMP, MEAN, ANNE – kaj izbrati za razvoj spletne aplikacije?

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Matjaž Kukar

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Tradicionalen razvoj spletnih aplikacij tipično poteka na osnovi skupin tehnologij, med katerimi je najpopularnejša skupina LAMP (Linux, Apache, MySQL, PHP) in njene različice. Z razvojem predvsem nerelacijskih podatkovnih baz in modernih skriptnih jezikov se pojavljajo tudi druge popularne skupine, npr. MEAN (MongoDB, Express, Angular, Node) in njena različica ANNE (Angular, Node, Neo4j, Express). Osvetlite bistvene razlike med tradicionalnimi in modernimi skupinami tehnologij za razvoj spletnih aplikacij, ter jih ovrednotite na praktičnem primeru. Opišite prednosti in slabosti posameznih skupin ter skušajte odgovoriti na vprašanje o vlogi skupin tehnologij v sodobnem razvoju spletnih aplikacij.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Tim Cestnik sem avtor diplomskega dela z naslovom:

LAMP, MEAN, ANNE – kaj izbrati za razvoj spletne aplikacije?

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Matjaža Kukarja,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 15. februarja 2016 Podpis avtorja:

Zahvaljujem se mentorju doc. dr. Matjažu Kukarju za vso pomoč in napotke pri nastajanju tega diplomskega dela ter staršem za podporo tekom študija.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
SPA	Single Page Application	aplikacija z eno stranjo
AJAX	Asynchronous JavaScript and XML	asinhroni JavaScript in XML
HTML	HyperText Markup Language	jezik za označevanje nadbesedila
CSS	Cascading Style Sheets	kaskadne stilske podloge
SQL	Structured Query Language	strukturirani povpraševalni jezik
API	application programming interface	vmesnik za programiranje aplikacij
JSON	JavaScript Object Notation	format za prenos objektov
CSV	Comma Separated Values	vrednosti, ločene z vejico

Povzetek

Naslov: LAMP, MEAN, ANNE – kaj izbrati za razvoj spletne aplikacije?

Namen diplomskega dela je predstaviti modernejši skupini tehnologij za razvoj spletnih aplikacij MEAN in ANNE v primerjavi s starejšo skupino tehnologij LAMP. Delo se osredotoča na tehnologije, ki sestavljajo te skupine, in na principe, po katerih delujejo. Poseben poudarek je na podatkovnih bazah. Opisane so prednosti in slabosti posameznih skupin tehnologij. V okviru diplomskega dela je bila z modernejšima skupinama tehnologij ANNE in MEAN razvita spletna aplikacija za iskanje prostorov in poti po stavbi Fakultete za računalništvo in informatiko v Ljubljani. Na tem primeru je bil prikazan razvoj spletne aplikacije na strani strežnika in odjemalca ter uporaba podatkovnih baz pri skupinah tehnologij ANNE in MEAN v primerjavi z LAMP. Na koncu diplomskega dela je ovrednoten smisel skupin tehnologij za razvoj spletnih aplikacij in opisana uporabniška izkušnja pri razvoju spletne aplikacije s posameznimi skupinami tehnologij. Ponujen je odgovor, katero skupino tehnologij uporabiti za razvoj spletne aplikacije.

Ključne besede: skupine tehnologij za razvoj spletnih aplikacij, LAMP, MEAN, ANNE, JavaScript.

Abstract

Title: LAMP, MEAN, ANNE - what to choose for Web development?

The purpose of thesis is to present and compare modern MEAN and ANNE web stacks with older LAMP stack. The main focus of this work is on the technologies used in these stacks and on the principles that these technologies are based on, with a special emphasis on databases. Pros and cons of each of the stack are evaluated. Web application for finding shortest paths in the Faculty of Computer and Information Science building in Ljubljana was developed using ANNE and MEAN stacks. Server-side development, client-side development and the use of database with ANNE and MEAN stacks was presented on this case compared to the use of LAMP stack. At the end, the logic of the use of web stacks was evaluated and user experience of using MEAN, ANNE and LAMP stacks was described. The answer to the question of which web stack to choose for web application development is given, recommending the use of MEAN and ANNE stacks.

Keywords: web stack, LAMP, MEAN, ANNE, JavaScript.

Poglavje 1

Uvod

Pri izdelavi spletnih aplikacij nam je danes na voljo mnogo različnih tehnologij. V splošnem moramo za izdelavo spletne aplikacije izbrati štiri elemente: operacijski sistem ali izvajalno okolje, spletni strežnik, programski jezik in podatkovno bazo.

Izbiramo lahko med mnogimi odprtokodnimi ali licenčnimi spletnimi strežniki, od katerih lahko mnogi tečejo na različnih operacijskih sistemih. Na voljo nam je tudi veliko število odprtokodnih in licenčnih podatkovnih baz. Te so lahko relacijske (SQL) ali nerelacijske (NoSQL), med nerelacijskimi pa lahko nadalje izbiramo med dokumentnimi, grafnimi, podatkovnimi bazami s pari ključ-vrednost in drugimi. Tudi med programskimi jeziki imamo široko izbiro. Nekateri danes najbolj popularni so JavaScript, Python, Java, PHP, Ruby in C++.

Edina omejitev pri izbiri teh osnovnih štirih elementov je ta, da se vsi med seboj podpirajo. Ker imajo predvsem odprtokodne tehnologije širok nabor podpore drugih tehnologij, imamo odprtih veliko možnosti.

A kljub temu so se pojavile t. i. skupine tehnologij za razvoj spletnih aplikacij (web stacks), ki ponujajo celostno, dobro dokumentirano rešitev za postavitev spletne aplikacije.

Skupine tehnologij za razvoj spletnih aplikacij v splošnem sestavljajo spletni strežnik, podatkovna baza, operacijski sistem oziroma izvajalno okolje ter

programski jezik. Elementi takšnih skupin se običajno razvijajo neodvisno drug od drugega, pogosto pa določene elemente uporabimo skupaj, tako pa nastanejo močna orodja za razvoj spletnih aplikacij.

Danes obstaja več uveljavljenih skupin tehnologij za razvoj spletnih aplikacij, njihova imena pa so običajno sestavljena iz prvih črk elementov, ki jih sestavljajo.

Največkrat uporabljena na svetovnem spletu je skupina tehnologij za razvoj spletnih aplikacij LAMP, ki obstaja že od leta 1996 in jo ponuja tudi večina spletnih gostovanj. A vsekakor ni edina. V zadnjih letih so veliko popularnost doživele modernejša skupina tehnologij za razvoj spletnih aplikacij MEAN in njene različice. Ena takih je skupina tehnologij za razvoj spletnih aplikacij ANNE.

Cilj tega diplomskega dela je raziskati in ovrednotiti novejši, modernejši skupini tehnologij MEAN in ANNE v primerjavi z na spletu največkrat uporabljeno skupino tehnologij LAMP.

Poleg raziskovanja in vrednotenja je cilj tega diplomskega dela tudi uporaba tehnologij MEAN in ANNE v praksi. S tema dvema skupinama tehnologij bo razvita sodobna spletna aplikacija, na podlagi večletnih izkušenj pa bo opisana tudi primerjava z razvojem sodobne spletne aplikacije s skupino tehnologij LAMP.

Prvi del tega diplomskega dela opisuje zgodovinski razvoj odprtokodnih tehnologij za razvoj spletnih aplikacij, sledi pa predstavitev sodobnih skupin tehnologih LAMP, MEAN in ANNE. Pregledu področja sledi prikaz praktičnega razvoja spletne aplikacije z uporabo predstavljenih tehnologij. V sklepnem delu ovrednotimo vse tri pristope razvoja spletne aplikacije, smisel uporabe skupin tehnologij za razvoj spletnih aplikacij ter ponudimo odgovor na vprašanje, kdaj uporabiti katero od izbranih treh skupin tehnologij.

Poglavje 2

Skupine tehnologij za razvoj spletnih aplikacij

Leta 1996 se je pojavila odprtokodna skupina tehnologij za razvoj spletnih aplikacij z imenom LAMP. Sestavljajo jo operacijski sistem Linux, Apache strežnik HTTP, podatkovna baza MySQL ter programski jezik PHP. LAMP je predstavljal idejo, da moramo na operacijski sistem gledati kot na integralni del aplikacije [1]. Komponente so bile razvite neodvisno, a filozofija in orodja so se razvijala z vednostjo, da ostale komponente obstajajo.

Popularnost skupine tehnologij LAMP je pričela naraščati med internetnim bumom v poznih devetdesetih letih [2]. Takrat so bila pri razvoju poslovnih spletnih strani največkrat uporabljena Microsoftova orodja in produkti podjetja Sun Microsystems. A zaradi nižjih cen, večje prilagodljivosti in hitrejših izboljšav komponent so kmalu mnoga podjetja pričela LAMP uporabljati tudi za razvoj večjih spletnih aplikacij. V začetku let 2000 so tako LAMP prevzela tudi nekatera večja podjetja kot so Google, Yahoo!, Lufthansa in Sabre Travel Network.

Premik proti LAMP-u se je torej deloma zgodil kot odziv na visoke cene komercialne programske opreme. Odprtokodna programska oprema je zastonj ali nizke cene, dodatni stroški pa nastanejo zaradi podpore in dodajanja funkcij. Razmerje med vsemi stroški pri LAMP-u in pri komercialnih ponu-

dnikih pa je sicer težko določiti, saj je odvisno od tega, katere strokovnjake ima podjetje, cene dela pogodbenikov, sredstev, ki so na voljo, in obstoječe infrastrukture.

Drug razlog za premik proti LAMP-u so tudi hitrejše izboljšave in popravki, kar je mogoče zato, ker so tehnologije odprtokodne. Lastniška programska oprema gre skozi počasnejši razvojni proces. Zagovorniki lastniške programske opreme pa pravijo, da to ponuja prednosti, saj gre za previden, koordiniran razvoj aplikacij, ki delujejo skupaj.

Lastniška programska oprema kupca pogosto tudi omejuje na izdelke le enega proizvajalca, kar je tudi eden izmed razlogov, da je odprtokodna skupina tehnologij LAMP postala tako popularna.

Razvijalci poslovnih aplikacij so imeli v preteklosti dvome v kvaliteto odprtokodne programske opreme in motilo jih je, ker ni bilo podpore za aplikacije, ki so jih ustvarili neodvisni razvijalci. Motilo jih je tudi, da bi lahko njihovi tekmeci njihove lastne dodatke odprtokodnim programskim opremam sami uporabili. Podjetja imajo sedaj lastne licence, ki to preprečujejo. Pojavila so se tudi podjetja, ki pomagajo postaviti in upravljati odprtokodne spletne aplikacije. Podjetja, ki ponujajo lastniško programsko opremo, so se na to odzvala tako, da so svojim nizkocenovnim orodjem za razvoj spletnih aplikacij dodala več funkcij.

2.1 Programski jeziki

Kmalu pa se je pojavila potreba po programskem jeziku, ki bi ga bilo mogoče pri razvoju spletnih aplikacij uporabiti na strani strežnika in odjemalca [3]. Razvijalec spletnih aplikacij mora poznati več področij, med drugimi podatkovne baze, varnost, programiranje na strani strežnika ter programiranje na strani odjemalca. Razvoj na strani odjemalca poteka v treh jezikih: HTML, CSS in JavaScript. Razvoj na strani strežnika pa poteka v programskih jezikih kot so PHP, Perl, Java, Python itd. Večina spletnih strani komunicira tudi s podatkovno bazo preko poizvedb, napisanih v poizvedovalnem pro-

gramskem jeziku, npr. SQL. To pomeni, da mora razvijalec spletnih strani poznati kar pet programskih jezikov – HTML, CSS, JavaScript, programski jezik na strani strežnika in poizvedovalni jezik. Ker dobro poznavanje petih različnih programskih jezikov, vsakega s svojo sintakso, presega sposobnosti enega programerja, je prišlo do tega, da sta pri razvoju na strani strežnika in odjemalca pričeli delati dve različni skupini razvijalcev, kar pa je pogosto vodilo do slabe usklajenosti pri razvoju aplikacij. Najuspešnejši poskus uporabe istega programskega jezika na obeh straneh razvoja je uporaba programskega jezika JavaScript.

Programski jezik JavaScript se je pod imenom Mocha pojavil leta 1995 [4]. Pri podjetju Netscape so želeli ustvariti lahek interpretirani programski jezik, ki bi bil primeren za neprofesionalne programerje. JavaScript se je pod imenom LiveScript pojavil skupaj s spletnim brskalnikom Netscape Navigator 2.0, pod imenom JavaScript pa je bil standardiziran [5] junija 1997. Od takrat je postal de facto jezik za razvoj na strani odjemalca. JavaScript so prevzeli tudi drugi razvijalci spletnih brskalnikov in vanj vložili veliko truda in denarja, zato je, kljub temu da je sprva veljal za programski jezik z malo zmogljivosti, ki ga uporabljajo zgolj amaterski razvijalci, postal programski jezik, primeren za kakršno koli splošno programersko nalogo. Postal je najpopularnejši programski jezik na svetovnem spletu in je edini jezik, ki ga podpirajo vsi glavni internetni brskalniki.

Podjetje Netscape je še istega leta, kot je izšel JavaScript, izdalo implementacijo jezika za programiranje na strani strežnika, a do večjega zanimanja za JavaScript na strani strežnika je prišlo šele sredi let 2000. Pomembnejši premik se je zgodil konec leta 2008, ko je skupaj s spletnim brskalnikom Chrome izšel tudi Googlov V8 JavaScript Engine [6]. Od takrat je izšlo več strežniških implementacij JavaScripta, ki za izvajanje kode uporabljajo V8.

Ena takih je Node.js [31], ki je bila ustvarjena leta 2009 in je odprtokodno izvajalno okolje, ki se v glavnem uporablja za razvoj spletnih aplikacij [7]. Node.js ponuja dogodkovno arhitekturo in API za asinhron V/I. Vsebuje tudi knjižnico, ki dovoljuje aplikacijam, da se obnašajo kot samo-

stojen spletni strežnik. Zaradi Node.js je postal programski jezik JavaScript močna alternativa programskim jezikom na strani strežnika. Ena najboljših zmožnosti Node.js je ta, da je veliko funkcionalnosti možno spraviti v zelo malo programske kode. Node je daleč najuspešnejši poskus implementacije JavaScripta na strani strežnika. Prevzela so ga tudi velika podjetja kot so Microsoft, Wallmart, PayPal in LinkedIn. Razvitih je bilo tudi ogromno dodatnih modulov za Node. Najpomembnejši izmed teh je Express.js [32], ki je ogrodje za strežnik za spletne aplikacije, s katerim je mogoče narediti spletne aplikacije z eno stranjo (SPA), z več stranmi in hibridne spletne aplikacije.

2.2 Podatkovne baze

Sočasno z razvojem Node.js pri strežnikih se je tudi pri podatkovnih bazah dogajal razvoj [12]. Pojavile so se nerelacijske podatkovne baze, ki za hranjenje podatkov ne uporabljajo tabel.

Take podatkovne baze hranijo podatke v različnih strukturah, npr. v dokumentih ali parih ključ-vrednost. So manj strukturirane, kar pogosto pomeni lažji razvoj. Nerelacijske podatkovne baze so pogosto tudi hitrejše, saj ne vsiljujejo tabelne strukture relacijskih baz.

Leta 2009 se je pojavila nerelacijska podatkovna baza MongoDB [34], ki prav tako uporablja V8, podatke pa hrani v dokumentih tipa Binary JavaScript Object Notation (BSON). MongoDB zato zelo preprosto bere in piše objekte iz JavaScripta.

2.3 Aplikacije AJAX in SPA

Z razširitvijo hitrega interneta so postale mogoče aplikacije AJAX (asinhroni JavaScript in XML) [10]. Takšne aplikacije obdelujejo zahtevke v ozadju, namesto da prihaja do nalaganja nove strani. Tako je sčasoma prišlo do pojava spletnih strani z eno stranjo (SPA — Single Page Application) [11], pri katerih pride do nalaganja ene same strani, vsi ostali zahtevki pa se obdelujejo

preko klicev AJAX. Leta 2009 se je pojavilo najbolj popularno ogrodje za spletne aplikacije z eno stranjo, AngularJS [14]. Uporablja pristop model-pogled-kontrolnik, poleg tega pa omogoča dvosmerno spajanje podatkov med pogledi in modeli, kar pomeni, da AngularJS vse samodejno usklajuje.

Kombinacija tehnologij MongoDB, Express.js, AngularJS in Node.js, s katerimi lahko ustvarimo spletno aplikacijo zgolj v programskih jezikih JavaScript, HTML in CSS, je postala tako popularna, da je dobila ime "MEAN stack"[3], iz nje pa je izpeljana tudi skupina tehnologij ANNE [13].

Poglavje 3

Sodobne skupine tehnologij za razvoj spletnih aplikacij

3.1 Skupina tehnologij LAMP

LAMP je okrajšava za skupino tehnologij za razvoj spletnih aplikacij, ki jo sestavlja odprtokodna programska oprema. To je operacijski sistem Linux, Apache strežnik HTTP, podatkovna baza MySQL ter programski jezik PHP.

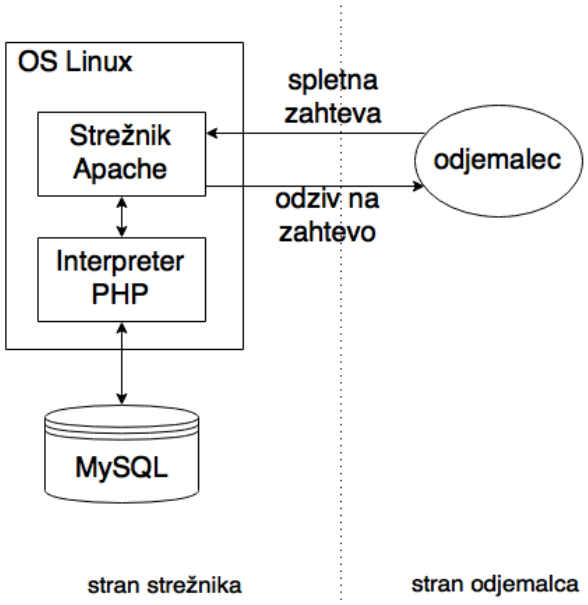
Slika 3.1 prikazuje diagram delovanja spletne aplikacije, narejene s skupino tehnologij LAMP. Odjemalec naredi zahtevo na spletni strežnik Apache, ki teče na operacijskem sistemu Linux. Zahteva se obdela tako, da interpreter PHP prevede kodo PHP, s katero dobimo podatke iz podatkovne baze MySQL, ki jih strežnik Apache posreduje odjemalcu.

Ime LAMP se pogosto uporablja tudi za različice osnovne skupine tehnologij LAMP s podatkovno bazo MariaDB namesto MySQL in programskima jezikoma Perl ali Python namesto PHP.

Skupina tehnologij LAMP je ena najbolj pogostih platform pri ponudnikih spletnega gostovanja.

Glavni konceptualni razliki med skupino tehnologij LAMP in sodobnejšima skupinama tehnologij ANNE in MEAN sta naslednji: LAMP na strani odjemalca ne zahteva specifične knjižnice JavaScript in je vezan na fizični

operacijski sistem (Linux), medtem ko ANNE in MEAN na strani odjemalca zahtevata knjižnico AngularJS in nista vezani na fizični operacijski sistem, ampak na izvajalno okolje Node.js.



Slika 3.1: Diagram delovanja spletne aplikacije, narejene s tehnologijami LAMP.

3.1.1 Linux

Linux je odprtokodni operacijski sistem, podoben operacijskim sistemom UNIX [16]. Glavna komponenta sistema je jedro Linux, na katerem temeljijo operacijski sistemi Linux. Jedro Linux je bilo prvič izdano leta 1991.

Čeprav je bil Linux sprva mišljen za uporabo na osebnih računalnikih, je postal vodilni operacijski sistem na strežnikih.

Obstaja več različnih distribucij Linuxa za uporabo na strežniku, ki jih sestavlja jedro Linux in zbirka programske opreme. Najbolj razširjena distribucija Linux na strežnikih je Debian, sledijo pa ji Ubuntu, CentOS, Red Hat, Gentoo, Fedora in SuSE [17].

Pogosti razlogi, ki se navajajo za tolikšno razširjenost Linuxa na strežnikih

[18], so predvsem odprtokodnost, stabilnost in varnost. Sistemi Linux lahko v primerjavi s sistemi Windows tečejo leta in leta, ne da bi se zrušili. Veliko bolje obdelujejo več procesov hkrati, ob spremembah konfiguracije pa običajno ni potrebe po ponovnem zagonu. Uporabniki samodejno nimajo administratorskih uporabniških dovoljenj, kar pomeni večjo modularnost sistema, to pa zagotavlja večjo varnost.

3.1.2 Apache

Apache je spletni strežnik HTTP, ki je igral ključno vlogo pri širjenju svetovnega spleta [19]. Vse od leta 1996 ostaja najpopularnejši spletni strežnik.

Apache se najpogosteje uporablja z operacijskimi sistemi Linux ali z drugimi sistemi, ki temeljijo na sistemih UNIX.

Spletni strežnik Apache je zgrajen modularno; mnoge funkcije so implementirane v modulih, ki razširjajo Apachejeve osnovne funkcionalnosti. To so moduli s podporo za programske jezike na strani strežnika, moduli za avtentikacijo, moduli za SSL in TLS, moduli za namestniški strežnik (proxy), modul za spreminjanje URL-ja in drugi.

Na strežniku Apache lahko strežemo več različnih domen z različno vsebino. Ker je Apache odprtokoden, ga je zelo preprosto prilagoditi različnim potrebam.

3.1.3 MySQL

MySQL je večnitni, večuporabniški relacijski sistem za upravljanje s podatkovnimi bazami, ki je prvič izšel leta 1995 [20]. MySQL je najbolj razširjen sistem za upravljanje s podatkovnimi bazami na spletu.

Podatkovne baze MySQL so relacijske, kar pomeni, da podatke hranijo v obliki tabel. Vrstice tabel predstavljajo entitete, stolpci tabel pa atribute entitet. Vsaka tabela predstavlja svoj entitetni tip.

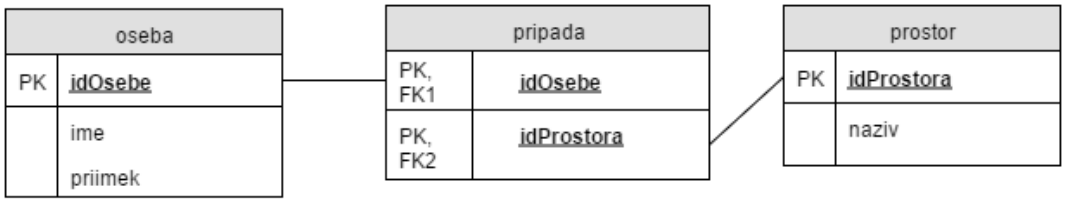
Tabele relacijskih podatkovnih baz so povezane z uporabo primarnih in tujih ključev. Primarni ključ je atribut tabele, ki enolično definira vsako

vrstico tabele. Tuji ključ je referenca na primarni ključ v drugi tabeli.

V relacijskih podatkovnih bazah lahko implementiramo več vrst povezav:

- 1:1: Vsak primarni ključ tabele je referenciran v največ enem zapisu v drugi tabeli.
- 1:n: Vsak primarni ključ tabele je referenciran v poljubnem številu zapisov v drugi tabeli.
- m:n: Vsak zapis v obeh tabelah je lahko povezan s poljubnim številom zapisov druge tabele. Relacijske podatkovne baze te povezave ne morejo implementirati neposredno; implementiramo jih z uporabo t. i. vmesne tabele.

Slika 3.2 prikazuje primer povezave m:n, implementirane v relacijski podatkovni bazi.



Slika 3.2: Primer povezave m:n v relacijski podatkovni bazi.

3.1.4 PHP

Programski jezik PHP je popularen splošen skriptni jezik, ki se uporablja predvsem za razvoj spletnih aplikacij na strani strežnika. Sintaksa je podobna jezikom Java, C in Perl [21].

PHP deluje na sistemih Linux, Unix, Microsoft Windows, Mac OS X, RISC OS in drugih. Podpira tudi mnoge spletne strežnike, kot sta Apache in IIS. Deluje lahko kot modul ali procesor CGI.

Ukaze PHP se lahko vključi direktno v dokument HTML, ne da bi klicali zunanjo datoteko za procesiranje podatkov. Koda PHP je zapisana med

posebna ukaza `<?php` in `?>`, ki nakazujeta začetek in konec kode PHP. Kodo PHP interpretira spletni strežnik z modulom za procesiranje PHP-ja, ki generira HTML zahtevane spletne strani. Odjemalec tako izvedene kode ne vidi, vendar vidi samo generirano kodo HTML.

Poleg kode HTML lahko generira tudi drugo kodo, na primer dokumente PDF ali filme Flash.

PHP podpira mnoge podatkovne baze. Povezava je mogoča s pripadajočimi knjižnicami ali knjižnico PDO (PHP Data Objects – podatkovni objekti PHP).

Spremenljive v PHP-ju so lahko tipa boolean, integer, float, string, array in object. Tipa spremenljivke ne določi programer, temveč jo procesor PHP določi ob začetku izvajanja glede na kontekst. Pred imeni spremenljivk stoji znak `$`. S funkcijo `define()` lahko definiramo tudi konstante, katerim se vrednost med izvajanjem skripte ne spremeni.

Obstaja več vnaprej definiranih spremenljivk, kot so `$_SERVER`, `$_GET`, `$_POST`, `$_SESSION` itd.

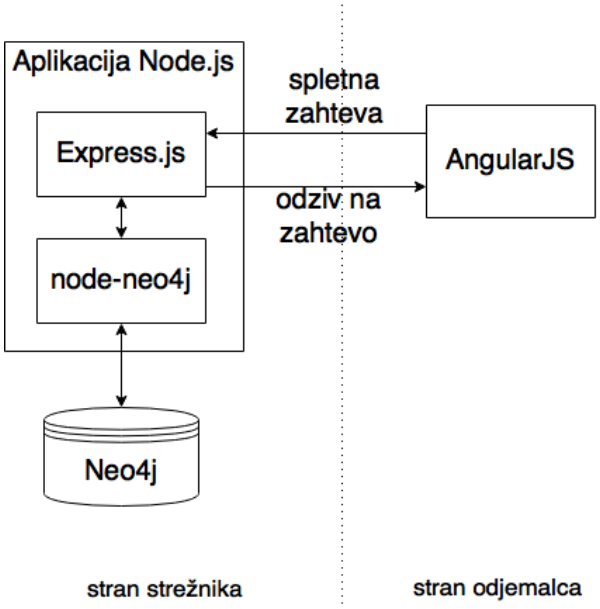
PHP vsebuje tudi funkcije za avtentikacijo, piškotke, spletne seje, nalaganje datotek, kompresijo, procesiranje kreditnih kartic, procesiranje slik, funkcije za protokole spletne pošte, matematične funkcije, procesiranje teksta, XML-ja in mnoge druge [22].

3.2 Skupini tehnologij ANNE in MEAN

Skupino tehnologij za razvoj spletnih aplikacij ANNE sestavljajo podatkovna baza Neo4j, ogrodje za spletni strežnik Express.js, ogrodje za spletne aplikacije z eno stranjo AngularJS in izvajalno okolje Node.js.

Slika 3.3 prikazuje diagram delovanja spletne aplikacije, narejene s skupino tehnologij ANNE. Aplikacija Node.js deluje kot samostojen spletni strežnik. Na zahtevo odjemalca naredi AngularJS spletno zahtevo na strežnik. S pomočjo ogrodja Express.js na strani strežnika se zahteva prične obdelovati. Knjižnica node-neo4j [23] za komunikacijo s podatkovno bazo Neo4j izvede

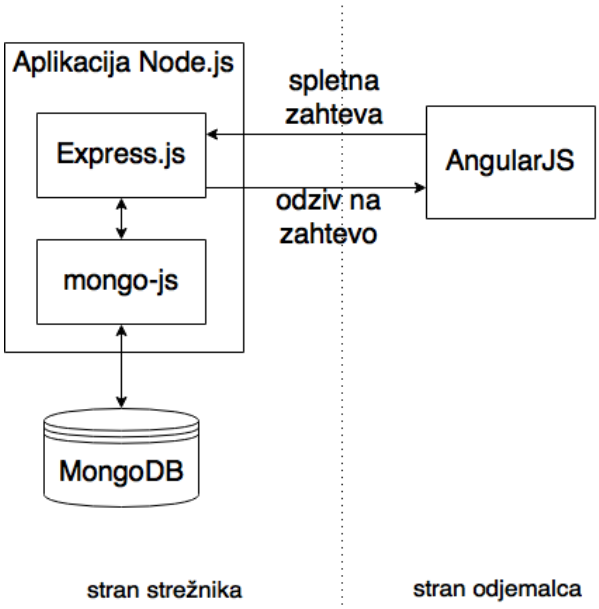
poizvedbo na podatkovno bazo. Podatkovna baza Neo4j vrne podatke, ki jih Express.js posreduje nazaj do AngularJS, ta pa jih prikaže odjemalcu.



Slika 3.3: Diagram delovanja spletne aplikacije, narejene s tehnologijami ANNE.

Skupina tehnologij za razvoj spletnih aplikacij MEAN je različica skupine ANNE, ki namesto grafne podatkovne baze Neo4j uporablja dokumentno podatkovno bazo MongoDB. Vsi ostali elementi so isti kot pri skupini ANNE.

Slika 3.4 prikazuje diagram delovanja spletne aplikacije, narejene s skupino tehnologij MEAN. Aplikacija Node.js deluje kot samostojen spletni strežnik. Na zahtevo odjemalca naredi AngularJS spletno zahtevo na strežnik. S pomočjo ogrodja Express.js na strani strežnika se zahteva prične obdelovati. Knjižnica mongojs [24] za komunikacijo s podatkovno bazo MongoDB izvede poizvedbo na podatkovno bazo. Podatkovna baza MongoDB vrne podatke, ki jih Express.js posreduje nazaj do AngularJS, ta pa jih prikaže odjemalcu.

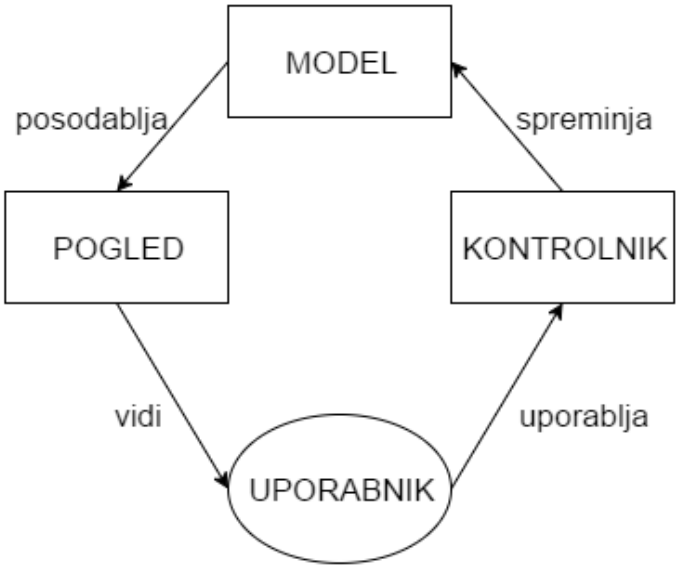


Slika 3.4: Diagram delovanja spletne aplikacije, narejene s tehnologijami MEAN.

3.2.1 AngularJS

AngularJS [14] je del skupin tehnologij ANNE in MEAN na strani odjemalca. Je odprtokodno ogrodje, ki rešuje probleme pri razvoju spletnih aplikacij z eno stranjo. AngularJS storitve, ki so običajno na strani strežnika, vpeljuje na stran odjemalca in s tem zmanjšuje breme na strežnik. Uporablja pristop model-pogled-kontrolnik (model-view-controller), s katerim loči komponente podatkov, prikaza in logike [25].

Model-pogled-kontrolnik [15] je arhitekturni pristop pri razvoju programske opreme, ki se uporablja predvsem za implementacijo uporabniških vmesnikov. Osrednji element pristopa je model, ki neposredno upravlja s podatki, logiko in pravili aplikacije ter je neodvisen od uporabniškega vmesnika. Pogled je predstavitev informacije uporabniku, kontrolnik pa sprejema vhodne podatke uporabnika in jih pretvarja v ukaze za model ali pogled. Slika 3.5 prikazuje diagram pristopa.



Slika 3.5: Diagram pristopa model-pogled-kontrolnik.

Ogrodje AngularJS razširja tradicionalen HTML za prikaz dinamičnih vsebin z uporabo dvosmerne vezave podatkov. Tako pride do avtomatične sinhronizacije med modeli in pogledi. Spremembe v modelu posodobijo pogled, spremembe v pogledu pa posodobijo model.

Deli kode HTML na strani odjemalca predstavljajo predlogo (template). Ti deli vsebujejo dodatna navodila (directives – direktive). AngularJS deluje tako, da najprej prebere stran HTML in direktive interpretira kot navodila, kako vezati vhodne in izhodne dele strani na model, ki ga predstavljajo standardne spremenljivke JavaScript. AngularJS model in predlogo združi v pogled.

Obstaja več direktiv, ki se pričnejo s predpono `ng-`. V elemente HTML jih lahko vključimo na naslednje načine: kot atribute značk HTML, kot značke HTML, kot komentarje HTML ali kot razrede CSS. Pomembnejše direktive so:

- `ng-app`: pove AngularJS, naj spletno stran prevede kot predlogo AngularJS spletne aplikacije. Običajno jo postavimo v značko `<html>`, ni pa to nujno.

- `ng-controller`: pove, kateri kontrolnik krmili delovanje elementa HTML. Elementu HTML dodamo lastnost `ng-controller='ime kontrolnika'`.
- `ng-model`: uporablja se za dvosmerno povezovanje podatkov in pove, kam se shrani vrednost elementa HTML. Elementu HTML dodamo lastnost `ng-model="izraz"`. Vsebina elementa je enaka vsebini spremenljivke `$scope.izraz`.
- `ng-repeat`: element HTML iterira čez celoten seznam podatkov, ki jih prejme.

Modul je zbiralnik, ki vsebuje različne gradnike. Obstaja več modulov, napisanih s strani AngularJS, ki imajo predpono `ng`, lahko pa ustvarimo tudi lastne module.

Izrazi AngularJS so v kodi HTML zapisani znotraj dvojnih zavitih oklepajev. Izraz AngularJS je lahko matematična funkcija, logična operacija, filter ali spremenljivka.

Kontrolnik je konstruktor, ki pripravi podatke in funkcije, ki bodo uporabljene v aplikaciji. `$scope` je posebna spremenljivka AngularJS, ki deluje kot povezava med kontrolnikom aplikacije in pogledom. Izraz `{{izraz}}` v predlogi bo AngularJS dopolnil tako, da bo v spremenljivki `$scope` poiskal lastnost `izraz`, tj. `$scope.izraz`.

Koda 3.1 prikazuje primer predloge HTML, ki uporablja modul `module` in kontrolnik `controller`, napisana v kodi 3.2. Direktiva `ng-repeat` napolni tabelo s podatki v `$scope.items`.

Koda 3.1: Primer predloge HTML.

```
<html ng-app="module">
<body ng-controller="controller">
<tr ng-repeat="item in items">
  <td>{{item.Name}}</td>
  <td>{{item.Type}}</td>
</tr>
</body>
</html>
```

Koda 3.2: Primer modula s kontrolnikom.

```
var app = angular.module('module', []);
app.controller('controller', ['$scope', function($scope) {
  $scope.items = [{"Name": "Item1", "Type": "Type1"},
                  {"Name": "Item2", "Type": "Type2"}];
}]);
```

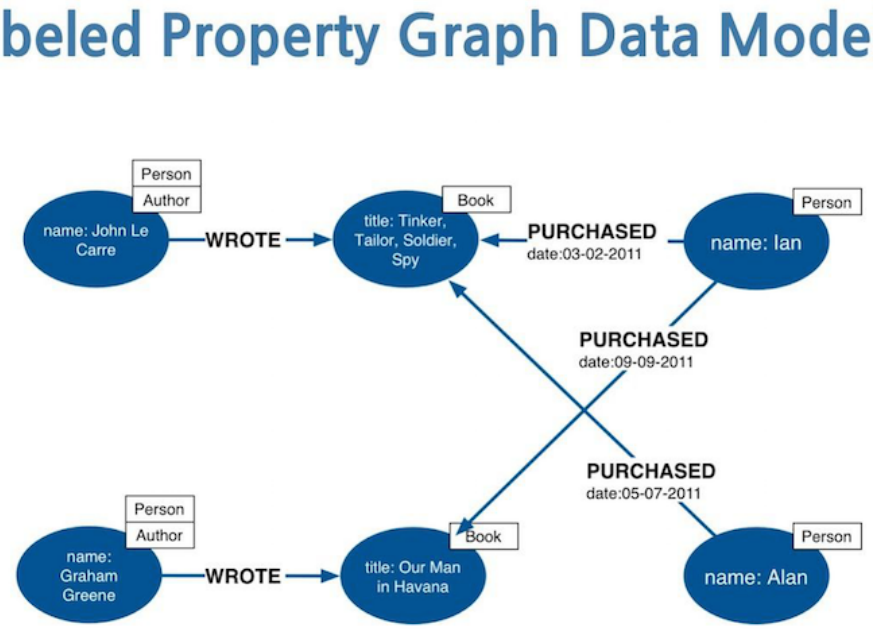
3.2.2 Neo4j

Neo4j [26] je odprtokodna grafna podatkovna baza, ki se je pričela razvijati leta 2003, javno dostopna pa je od leta 2007. Implementirana je v programskih jezikih Java in Scala, programom v drugih programskih jezikih pa je dostopna z uporabo poizvedovalnega jezika Cypher. Neo4j danes uporablja na sto tisoče podjetij, uporabljajo pa jo za vse od znanstvenih raziskav, iskanja poti, upravljanja omrežij, vodenja projektov do socialnih omrežij, priporočilnih sistemov in zmenkarij.

Neo4j podatke hrani v obliki označenega grafa z atributi [27]. Ta vsebuje povezana vozlišča, ki vsebujejo poljubno število atributov, ti pa so predstavljeni v obliki parov ključ-vrednost. Vozlišča so lahko označena z oznakami, ki opisujejo njihovo funkcijo znotraj domene ali pa vsebujejo metapodatke. Vsako vozlišče grafa predstavlja entiteto, vsaka povezava pa odnos med entitetama. Povezava ima vedno smer, tip, začetno vozlišče in končno vozlišče. Vsebuje lahko tudi poljubne atribute. Običajno so ti atributi kvantitativni – teža, cena, razdalja itd. Med dvema vozliščema lahko obstaja poljubno število povezav poljubnih tipov. Kljub temu da so vse povezave usmerjene, se lahko po grafu sprehajamo ne glede na smer povezav.

Povezave so prioriteta grafnih podatkovnih baz. Z njimi lahko tako modeliramo kakršnekoli podatke, ki jih definirajo povezave.

Na sliki 3.6 vidimo primer podatkov v označenem grafu z atributi. Graf vsebuje šest vozlišč. Dve vozlišči z oznako "Person" in dve z oznakama "Person" in "Author" vsebujeta atribut "name", dve z oznako "Book" pa atribut "title". Med vozlišči obstaja pet povezav. Dve tipa "Wrote" sta brez



Slika 3.6: Primer označenega grafa z atributi [28].

atributa, tri tipa "Purchased" pa imata atribut "date".

Če Neo4j primerjamo z relacijskimi podatkovnimi bazami, moramo pri relacijskih podatkovnih bazah najprej ustvariti logični model, nato pa ga pretvoriti v tabelarični, fizični model [29]. Pri grafni podatkovni bazi to ni potrebno, saj ne potrebujemo nobenih primarnih in tujih ključev; povezavo med dvema entitetama predstavlja kar povezava med vozliščema v grafu. Relacijske podatkovne baze imajo povezave realizirane z referencami na primarne ključe v tujem ključu. Pri poizvedovanju zato pride do dragih samodejnih stičnih operacij `JOIN`. Pri grafni podatkovni bazi vsako vozlišče vsebuje seznam fizičnih povezav z drugimi vozlišči, ki ga baza uporabi za direkten dostop do povezanih vozlišč brez uporabe dragih operacij iskanja.

Neo4j za poizvedovanje uporablja poizvedovalni jezik Cypher [30]. Cypher je deklarativen jezik, narejen po vzoru SQL-a, uporablja pa se ga za opisovanje vzorcev v grafu. Z njim opišemo, kaj želimo izbrati, vstaviti, posodobiti ali izbrisati iz grafne podatkovne baze, ne da bi opisali, kako to narediti.

Pomembnejša stavka poizvedovalnega jezika Cypher za branje iz podatkovne baze sta `MATCH` in `WHERE`. `MATCH` se uporablja za zapis vzorcev, ki jih iščemo v grafu, `WHERE` pa za dodajanje omejitev ali predikatov in se vedno uporablja v paru z `MATCH`. S stavkom `RETURN` povemo, kaj naj poizvedba vrne.

Pomembnejši stavki za pisanje v podatkovno bazo so `CREATE`, ki ustvari vozlišče ali povezavo, `MERGE`, ki poišče, če vzorec v grafu že obstaja in če ne obstaja, ustvari novega, `SET`, ki spremeni vrednosti atributov, `DELETE`, ki izbriše vozlišče ali povezavo, in `REMOVE`, ki izbriše atribut ali oznako vozlišča.

Koda 3.3 prikazuje primer poizvedbe Cypher s stavki `CREATE` in `MERGE`. Poizvedba ustvari dve vozlišči z oznako `Person` in `Department` in ju poveže s povezavo tipa `WORKS_IN`. Prvi stavek `MERGE` ne naredi ničesar, saj vozlišče s tako oznako in atributi že obstaja, drugi stavek `MERGE` pa ustvari novo vozlišče, saj vozlišče s tako oznako in atributi še ne obstaja.

Koda 3.3: Primer poizvedbe s stavkoma CREATE in MERGE.

```
CREATE (john: Person {FirstName: 'John', LastName: 'Doe'})
CREATE (it: Department {Name: 'IT Department'})
CREATE (john)-[:WORKS_IN]->(it)
MERGE (p: Person {FirstName: 'John', LastName: 'Doe'})
MERGE (p: Person {FirstName: 'Jane', LastName: 'Doe'})
```

Poizvedba v kodi 3.4 vrne vse osebe, ki delajo v oddelku "IT Department".

Koda 3.4: Primer poizvedbe s stavkoma MATCH in WHERE.

```
MATCH (p: Person), (d: Department)
WHERE d.Name = "IT Department"
RETURN p
```

Poizvedba v kodi 3.5 spremeni atribut `LastName` vozlišča z oznako `Person` in ID-jem 24.

Koda 3.5: Primer poizvedbe s stavkom SET.

```
MATCH (p: Person)
WHERE Id(p) = 24
SET p.LastName = "Carter"
RETURN p
```

3.2.3 Node.js

Node.js [31] je odprtokodno izvajalno okolje za razvoj spletnih aplikacij na delu strežnika. Ustvarjeno in izdano za operacijski sistem Linux je bilo leta 2009.

Zgrajeno je na Chromovem V8 JavaScript Engine in uporablja dogodkoven, neblokirajoč vhodno-izhodni model [7]. Node.js deluje na operacijskih sistemih OS X, Microsoft Windows, Linux, FreeBSD, NonStop OS, IBM AIX, IBM System z in IBM i. Večina osnovnih modulov je napisanih v JavaScriptu. Vsebuje vgrajeno knjižnico, ki omogoča, da se aplikacije obnašajo kot samostojni spletni strežniki.

Node.js ne smemo zamenjevati za splošno namenski spletni strežnik. Node.js je platforma za spletne aplikacije, ki zahteve HTTP obdelujejo asinhrono, ne more pa na istem strežniku servirati več različnih spletnih strani, napisanih v različnih programskih jezikih. Za to nalogo še vedno potrebujemo splošno namenski spletni strežnik. To, da aplikacija Node.js teče za splošno namenskim spletnim strežnikom, običajno rešujemo s pristopom "reverse proxy"[8]. To je obratno delovanje namestniškega strežnika – ko pride zahteva do splošno namenskega spletnega strežnika, jo ta posreduje Node.js, ta zahtevo sprocesira in jo pošlje nazaj splošno namenskemu spletnemu strežniku, ki jo prikaže uporabniku.

Leta 2011 je bil izdan upravljalec paketov za knjižnico Node.js z imenom npm [9], ki danes vsebuje več kot 230.000 paketov. Večina paketov je modulov, ki se jih v aplikacijo Node.js lahko naloži s klicem funkcije `require()`. To so različna ogrodja, ki pohitrijo razvoj spletne aplikacije na strani strežnika, knjižnice za povezavo s podatkovnimi bazami, knjižnice za razhroščevanje,

vodenje dnevnika in druge.

Z Node.js je mogoče ustvariti spletni strežnik z uporabo zbirke modulov in programskega jezika JavaScript. Izdelavo aplikacij je mogoče pohitriti z uporabo ogrodij kot sta Express.js in Socet.IO.

Node.js implementira dogodkovno programiranje v JavaScriptu, kar pomeni, da se nov ukaz ne izvrši šele, ko se predhodna vhodno-izhodna operacija konča, temveč se ukazi izvajajo naprej, ko se V/I operacija konča, pa se izvede funkcija povratnega klica.

Funkcija povratnega klica (callback) je funkcija, ki se izvrši, ko se V/I operacija dokonča ali pri izvajanju V/I operacije pride do napake. Ko pride do zahteve po V/I operaciji, se zahtevi priključi tudi funkcijo povratnega klica. Ukazi se izvajajo naprej, ko se V/I operacija konča, pa se izvede priključena funkcija povratnega klica.

Primer funkcije povratnega klica je prikazan v kodi 3.6. Funkcija `operacija()` prejme en parameter in funkcijo povratnega klica. Definiramo jo tako, da najprej napišemo kodo za izvedbo operacije, ko se ta izvrši, kličemo funkcijo povratnega klica. Ob klicu funkcije `operacija()` podamo parameter in funkcijo povratnega klica. Ko se funkcija `operacija()` prične izvrševati, ostali ukazi tečejo naprej, ko se funkcija izvrši, se pojavi obvestilo "Operacija končana".

Koda 3.6: Primer funkcije povratnega klica.

```
function operacija(parameter, callback) {
  // V/I operacija
  callback();
}

operacija('/locations', function() {
  alert('Operacija koncana.');
});
```

3.2.4 Express.js

Express.js [32] je del skupin tehnologij ANNE in MEAN na strani strežnika. Je ogrodje za vzpostavitev strežnika za spletne aplikacije na platformi Node.js.

S pomočjo ogrodja Express.js lahko na platformi Node.js spletni strežnik vzpostavimo v le nekaj vrsticah kode. Express.js ponuja funkcije, s katerimi povemo, kako se aplikacija odziva na zahteve odjemalca, podane s potjo ali URI (Uniform Resource Identifier) ter metodo za zahtevo HTTP.

Zahteve odjemalca z uporabo Express.js definiramo na način `aplikacija.METODA(POT, FUNKCIJA)` [33], kjer je:

- aplikacija: instanca express.
- METODA: metoda za zahtevo HTTP, ki je lahko tipa get, post, put, head, delete, options, trace, copy, lock, mkcol, move, purge, propfind, proppatch, unlock, report, mkactivity, checkout, merge, m-search, notify, subscribe, unsubscribe, patch, search in connect.
- POT: pot na strežniku.
- FUNKCIJA: funkcija, ki se izvrši ob zahtevi.

Koda 3.7 prikazuje primer definicije zahteve HTTP `GET` na domačo stran. Odzovemo se s stavkom "Hello World!"

Koda 3.7: Definicija zahteve HTTP GET na domačo stran.

```
app.get('/', function (req, res) {
  res.send('Hello World!');
});
```

3.2.5 MongoDB

MongoDB [34] je odprtokodna dokumentna podatkovna baza, prvič razvita leta 2007. Danes je četrti najpopularnejši sistem za upravljanje s podatkovnimi bazami in je najpopularnejša dokumentna podatkovna baza na spletu [35].

MongoDB podatkov ne hrani v tabelah kot relacijske podatkovne baze, temveč v dokumentih tipa Binary JavaScript Object Notation (BSON) [36]. BSON je binarna predstavitev tipa JSON z dodatnimi informacijami. MongoDB zato zelo preprosto bere in piše objekte iz JavaScripta.

Dokumenti vsebujejo pare polje-vrednost. Vrednost je lahko katerega koli tipa BSON, na primer `String`, `Double`, `Array`, lahko pa je tudi drug dokument ali seznam dokumentov.

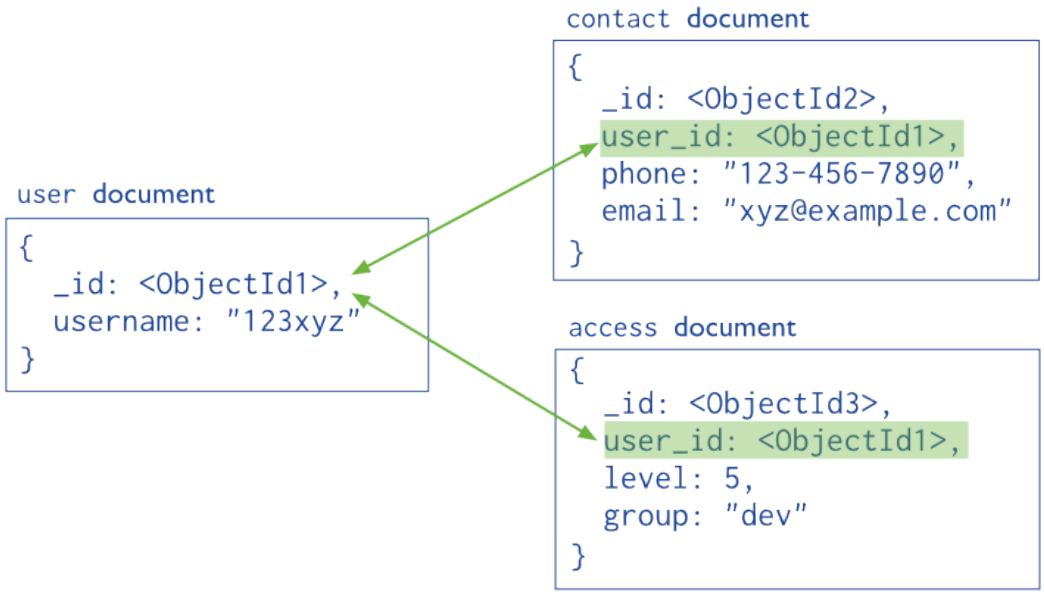
Podatkovna baza hrani vse dokumente v zbirkah (collections). Zbirka je skupina sorodnih dokumentov, ki si deli skupne indekse. Zbirke predstavljajo to, kar v relacijskih podatkovnih bazah predstavljajo tabele. Dokumenti predstavljajo to, kar vrstice znotraj tabel, polja dokumentov BSON pa to, kar stolpci tabel, z razliko, da lahko vsak dokument vsebuje poljuben nabor polj BSON. Ekvivalent primarnega ključa je polje `_id`, ki ga vsebuje vsak dokument baze. Lahko ga sami definiramo ob kreiranju dokumenta, če ga ne definiramo, ga podatkovna baza kreira samodejno v obliki objekta `ObjectId`.

Kljub temu da MongoDB ne ponuja nikakršnih povezav med dokumenti, lahko dokumente povezujemo sami z uporabo referenc, kot kaže slika 3.7.

Poizvedbe pri MongoDB uporabljajo notacijo JavaScript in se izvedejo nad določeno zbirko. Poizvedbe za branje podajo kriterije ali pogoje, ki identificirajo dokumente, ki jih MongoDB vrne odjemalcu. Izvedejo se s funkcijo `find()`. Dodamo lahko tudi projekcijo, ki pove, katera polja naj vrne poizvedba, in modifikatorje, s katerimi omejimo število rezultatov, določimo vrstni red itd.

Poizvedbe za spreminjanje podatkov ustvarijo, posodobijo ali izbrišejo dokumente. Izvedejo se s funkcijami `insert()`, `update()` in `delete()`. Izvajajo se nad določeno zbirko. Pri posodabljanju in brisanju dokumentov podamo kriterije za izbiro dokumenta, ki ga bomo posodobili ali izbrisali.

Za kriterije za izbiro obstajajo primerjalni operatorji (`$gt`, `$eq`, `$in` ...), logični operatorji (`$or`, `$and`, `not` ...), elementni operatorji, operatorji vrednotenja, geoprostorski operatorji, bitni operatorji, operatorji seznamov in komentarji.



Slika 3.7: Primer povezovanja dokumentov.

Obstaja tudi več operatorjev za posodabljanje, kot so `$set`, ki nastavi novo vrednost, `$rename`, ki spremeni ime polja, in drugi.

Poizvedbe lahko vsebujejo tudi uporabniško definirane funkcije JavaScript. Kompleksnejše poizvedbe rešujemo sami z višje nivojskimi programskimi jeziki.

Primer poizvedbe v kodi 3.8 dokumentu v zbirki `departments` z ID-jem 100 spremeni atribut `Name`.

Koda 3.8: Primer poizvedbe v MongoDB.

```
db.departments.update(
  { _id: 100 },
  { $set:
    { Name: "IT Department" }
  }
)
```

Poglavje 4

Spletna aplikacija FRI Map

Za potrebe diplomske naloge je bila s pomočjo skupin tehnologij za razvoj spletnih aplikacij ANNE in MEAN ustvarjena spletna aplikacija FRI Map. Aplikacija obiskovalcem stavbe Fakultete za računalništvo in informatiko (FRI) pomaga poiskati, kje se nahaja določen prostor, v katerih prostorih iskati določenega zaposlenega ter poišče najkrajšo pot med enim in drugim prostorom.

4.1 Osnovne funkcionalnosti

Spletna aplikacija FRI Map vsebuje dve dostopni točki – prvo za uporabnika, v kateri aplikacija prikaže načrt stavbe Fakultete za računalništvo in informatiko in omogoča iskanje prostorov, osebja in najkrajših poti, ter drugo za administratorja, ki omogoča dodajanje in brisanje zaposlenih in njihovih pripadnosti prostorom stavbe.

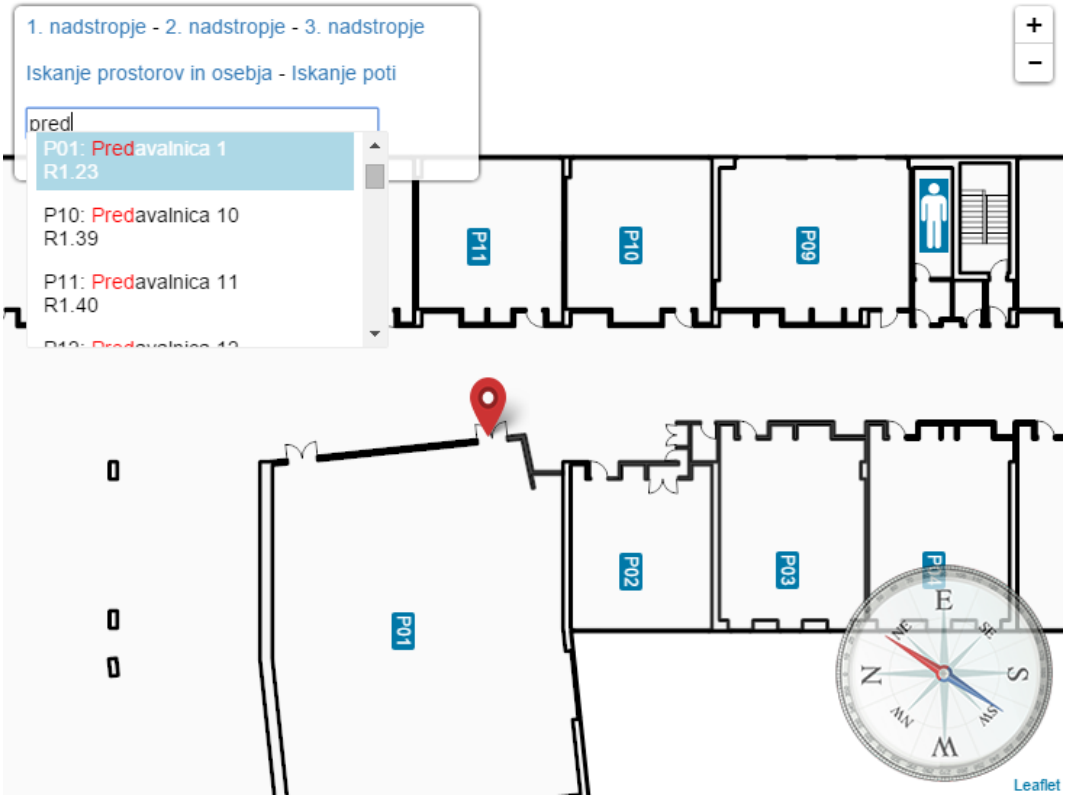
4.1.1 Uporabniška dostopna točka

Uporabniška dostopna točka uporabniku prikaže načrt prvega nadstropja stavbe Fakultete za računalništvo in informatiko. Z izbiro v izbirniku nadstropja lahko preklaplja med prikazi vseh treh nadstropij. Načrt lahko uporabnik poveča ali pomanjša v petih stopnjah povečave in se z vlečenjem

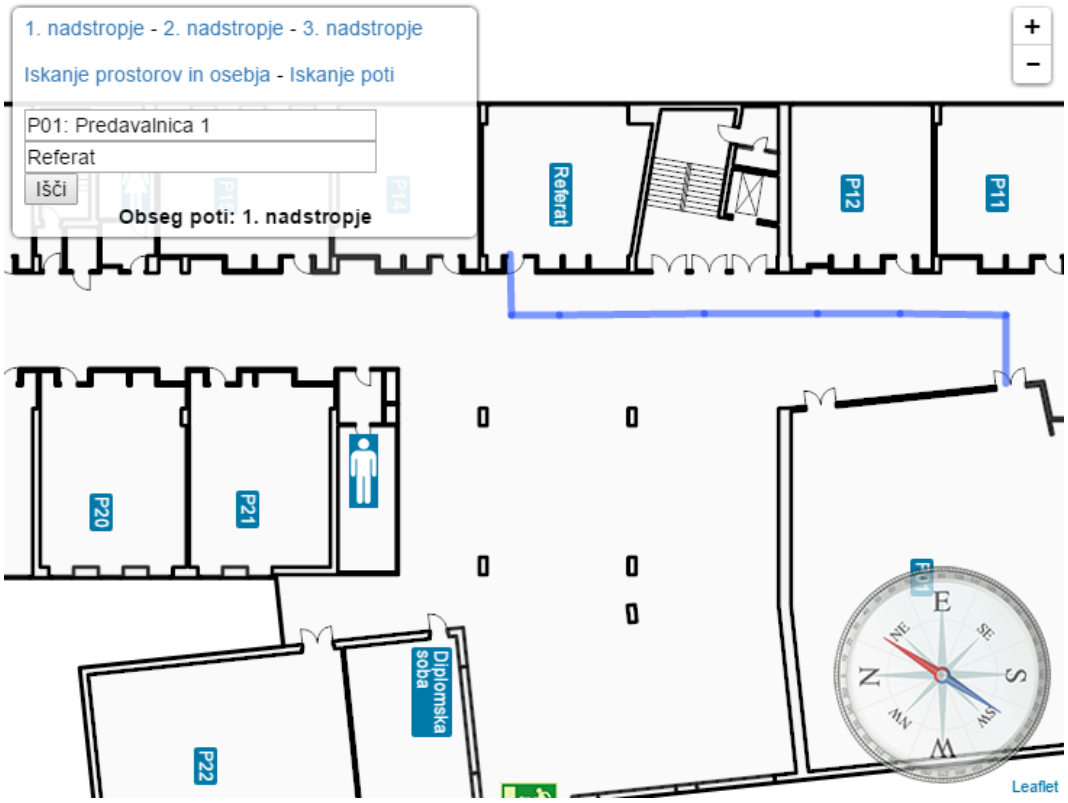
pomika po načrtu.

Spletna aplikacija vsebuje iskalno polje 'Išči prostor ali osebje na FRI'. S pričetkom pisanja imena prostora ali imena zaposlenega v to polje s funkcijo samodokončaj se uporabniku prikažejo predlogi. Z izbiro predloga in klikom na gumb 'Išči' se središče prikazanega načrta pomakne na vhod v izbran prostor oziroma prostor, v katerem se nahaja izbran zaposleni, in na načrtu se pojavi žebljiček, ki označuje vhod v prostor.

Aplikacija vsebuje tudi iskalnik za iskanje najkrajše poti med dvema prostoroma. Z izbiro začetne in končne točke ter klikom na gumb 'Išči' se na načrtu izriše najkrajša pot od začetne do končne točke. Aplikacija uporabniku izpiše tudi, katera nadstropja obsega iskana pot.



Slika 4.1: Iskanje prostorov ali zaposlenih.



Slika 4.2: Iskanje najkrajše poti.

4.1.2 Administratorjeva dostopna točka

Administratorjeva dostopna točka je dostopna preko naslova `/adm`. Zaščitena je s prijavo s parom uporabniško ime-geslo. Uporablja se za posodabljanje podatkov o zaposlenih in njihovi pripadnosti prostorom. Ob uspešni prijavi se administratorju prikaže možnost uvoza seznama profesorjev, možnost uvoza seznama zaposlenih v laboratorijih, možnost izbrisa vseh podatkov o zaposlenih ter vmesnik za ročno urejanje zaposlenih.

Administrator lahko uvozi seznam profesorjev v obliki datoteke CSV strukture `Ime;Priimek;Kabinet;Laboratorij`. Aplikacija prebere datoteko CSV in ustvari ustrezne vnose o zaposlenih in njihovi pripadnosti prostorom. Novi podatki se dodajo obstoječim podatkom v podatkovni bazi; če želimo

prejšnje podatke pobrisati, moramo pred uvozom novih uporabiti funkcijo za izbris vseh podatkov o zaposlenih. Koda 4.1 prikazuje primer datoteke CSV za uvoz profesorjev. Polji kabinet in laboratorij nista obvezni.

Koda 4.1: Primer datoteke CSV za uvoz profesorjev.

```
Danijel;Skočaj;R2.57;LUVSS
Aleš;Leonardis;R3.07;LUVSS
Marjan;Krisper;R2.35;
Saša;Divjak;R2.39;
```

Seznam zaposlenih v laboratorijih mora biti v obliki datoteke CSV strukture `Laboratorij;Vodja;Zaposleni`. Atribut `Zaposleni` vsebuje seznam imen in priimkov zaposlenih, ločen z vejico. Aplikacija prebere datoteko CSV in ustvari ustrezne vnose o zaposlenih in njihovi pripadnosti prostorom. Koda 4.2 prikazuje primer datoteke CSV za uvoz zaposlenih v laboratorijih.

Koda 4.2: Primer datoteke CSV za uvoz zaposlenih v laboratorijih.

```
BIOLAB;Blaž Zupan;Janez Demšar, Tomaž Curk, Jure Žbontar,
Marko Toplak, Miha Štajdohar, Aleš Erjavec, Anže Starič,
Tomaž Hočevar, Marinka Žitnik, Goran Bobojević,
Niko Colnerič, Jernej Ule, Martin Vuk, Lan Žagar,
Andrej Čopar, Martin Stražar

LALG;Borut Robič;Tomaž Dobravec, Boštjan Slivnik,
Jurij Mihelič, Uroš Čibej

LASPP;Uroš Lotrič;Branko Šter, Nejc Ilc, Davor Sluga,
Tom Vodopivec
```

Vmesnik za ročno urejanje zaposlenih omogoča dodajanje zaposlenega, urejanje zaposlenega, brisanje zaposlenega, dodajanje zaposlenega v prostor in brisanje zaposlenega iz prostora.

Slika 4.3: Administratorjeva dostopna točka.

4.2 Struktura aplikacije

Struktura naslovov spletne aplikacije FRI Map je enaka pri izdelavi s skupinama tehnologij ANNE ali MEAN in je sledeča:

- `/adm`: direktorij, ki vsebuje datoteke za administratorjevo vstopno točko, ki se izvajajo na strani odjemalca.
  - `/scripts`: direktorij, ki vsebuje datoteke JavaScript, uporabljene v aplikaciji.
  - `index.html`: datoteka s kodo HTML.

  - `app.js`: programska koda JavaScript aplikacije na strani odjemalca, uporabljena v datoteki `index.html`.

- `/node_modules`: direktorij, ki vsebuje v aplikaciji uporabljene module za Node.js, ki se izvajajo na strani strežnika.

- `/public`: direktorij, ki vsebuje datoteke za uporabniško dostopno točko, ki se izvajajo na strani odjemalca.

  - `/tiles`: direktorij, ki vsebuje načrt stavbe FRI v obliki ploščic velikosti 256px X 256px tipa PNG.
  - `/css`: direktorij, ki vsebuje datoteke CSS, uporabljene v aplikaciji.
  - `/scripts`: direktorij, ki vsebuje datoteke JavaScript, uporabljene v aplikaciji.
  - `/images`: direktorij, ki vsebuje slike, uporabljene v aplikaciji.
  - `index.html`: datoteka s kodo HTML.
  - `app.js`: programska koda JavaScript aplikacije na strani odjemalca, uporabljena v datoteki `index.html`.

- `server.js`: datoteka, ki vsebuje programsko kodo JavaScript, ki se izvaja na strani strežnika.

4.3 Aplikacija na strani odjemalca

Spletno aplikacijo FRI Map smo zasnovali tako, da je aplikacija na strani odjemalca enaka ob uporabi skupin tehnologij ANNE ali MEAN, saj smo podatke, ki jih aplikacija potrebuje od strežnika, pri zasnovi z ANNE in MEAN v formatu JSON vračali v popolnoma enaki obliki.

4.3.1 Vezava podatkov z uporabo AngularJS

V aplikaciji AngularJS pri uporabniški dostopni točki smo najprej ustvarili lasten modul z imenom `app`, v katerem smo uporabili modul `angucomplete-alt`,

s katerim smo ustvarili iskalnik prostorov in zaposlenih s funkcijo samodokončaj, kot prikazuje koda 4.3.

Koda 4.3: Modul AngularJS.

```
var app = angular.module('app', ["angucomplete-alt"]);
```

Ustvarili smo tudi kontrolnik z imenom `MainController`, ki je pripravil podatke, ki jih bomo uporabili v aplikaciji. Koda 4.4 prikazuje del, v katerem smo pripravili podatke za iskalnik prostorov. Uporabili smo storitvi `$scope` za vezavo podatkov in `$http` za izvedbo zahteve HTTP. Podatke smo v formatu JSON dobili preko zahteve HTTP tipa `GET` na naslov `/locations`.

Koda 4.4: Kontrolnik AngularJS.

```
app.controller('MainController', ['$scope', '$http',
function MainController($scope, $http) {
  $http.get('/locations').success(function(response) {
    $scope.locations = response;
  });
}]);
```

V kodi HTML smo znački `<html>` dodali direktivo `ng-app="app"`, s katero smo povedali, naj se stran prevede kot predloga AngularJS. Zanački `<body>` smo dodali direktivo `ng-controller="MainController"` in s tem povedali, naj se napisani kontrolnik uporabi v telesu kode HTML. Nato smo ustvarili iskalnik s funkcijo samodokončaj, na katerega smo pod atributom `local-data` privezali podatke v spremenljivki `$scope.locations`, kot prikazuje koda 4.5.

Koda 4.5: Iskalnik angucomplete-alt.

```
<div angucomplete-alt id="ex1"
  placeholder="{{ 'SEARCHLABEL' | translate }}"
  maxlength="50"
  pause="100"
  selected-object="selectedLocation"
  local-data="locations"
  search-fields="Display"
  title-field="Display"
  description-field="Description"
  minlength="1"
  input-class="form-control-small"
  match-class="highlight">
</div>
```

V aplikaciji za administratorja smo v spremenljivko `$scope.persons` shranili seznam vseh zaposlenih v formatu JSON, ki smo ga dobili z zahtevo HTTP tipa `GET` na naslovu `/persons`. V predlogi HTML smo nato v znački `<tr>` uporabili direktivo `ng-repeat`, ki je tabelo napolnila s podatki v spremenljivki `$scope.persons`, kot prikazuje koda 4.6.

Koda 4.6: HTML z direktivo ng-repeat.

```
<tr ng-repeat="person in persons">
  <td>{{person.FirstName}}</td>
  <td>{{person.LastName}}</td>
</tr>
```

Za dodajanje novega zaposlenega smo v kodi HTML vnosnima poljema `<input>` za ime in priimek dodali direktivo `ng-model="person.FirstName"` ter `ng-model="person.LastName"`, s čimer smo povedali, naj se sprememba v teh dveh poljih v modelu odraža v spremenljivki `$scope.person`. To spremenljivko smo nato preko zahteve HTTP tipa `POST` posredovali strežniku, ki je podatke shranil v podatkovno bazo, kot prikazuje koda 4.7.

Koda 4.7: Zahteva HTTP tipa POST v AngularJS.

```
$scope.addPerson = function() {
  $http.post('/adm/persons', $scope.person)
  .success(function(response) {
    refresh();
  });
};
```

4.3.2 Prikaz načrta

Načrt stavbe Fakultete za računalništvo in informatiko in graf poti po stavbi v obliki OpenStreetMap smo dobili iz diplomskega dela Robina Emeršiča z naslovom Mobilna aplikacija za prikaz prostorov in načrtovanje poti v stavbah [37].

OpenStreetMap (OSM) je zapis zemljevida v XML, sestavljen iz vozlišč, poti, relacij in oznak. Vsako vozlišče ima koordinate v standardu WGS84. Načrt v obliki OSM smo za uporabo v spletni aplikaciji s programom Maperative [38] pretvorili v obliko ploščic (tiles). To so sličice PNG velikosti 256px X 256px. Ustvarili smo ploščice v povečavah od 19 do 23, tj. v merilih od 1:1000 do 1:62,5.

Graf poti v obliki OSM vsebuje vozlišča s koordinatami, ki predstavljajo lokacije v stavbi FRI, in povezave med vozlišči, ki predstavljajo poti med sosednjima lokacijama. Slika 4.4 prikazuje grafični prikaz grafa poti prvega nadstropja v OSM. Graf smo ustrezno pretvorili za zapis v podatkovnih bazah Neo4j in MongoDB.

Za prikaz načrta stavbe Fakultete za računalništvo in informatiko in risanje po načrtu smo uporabili knjižnico Leaflet [39].

Leaflet je odprtokodna knjižnica JavaScript, prvič izdana leta 2011, ki se uporablja za prikaz zemljevidov v spletnih aplikacijah. Podpira HTML5 in CSS3 ter večino mobilnih in namiznih platform.

Leaflet na element HTML `<div>` veže zemljevid, ki mu nato lahko dodajamo več plasti. Zemljevid mora biti v obliki ploščic (tiles).

Lokacije na zemljevidu podajamo kot geografsko dolžino in širino v decimalnih stopinjah.

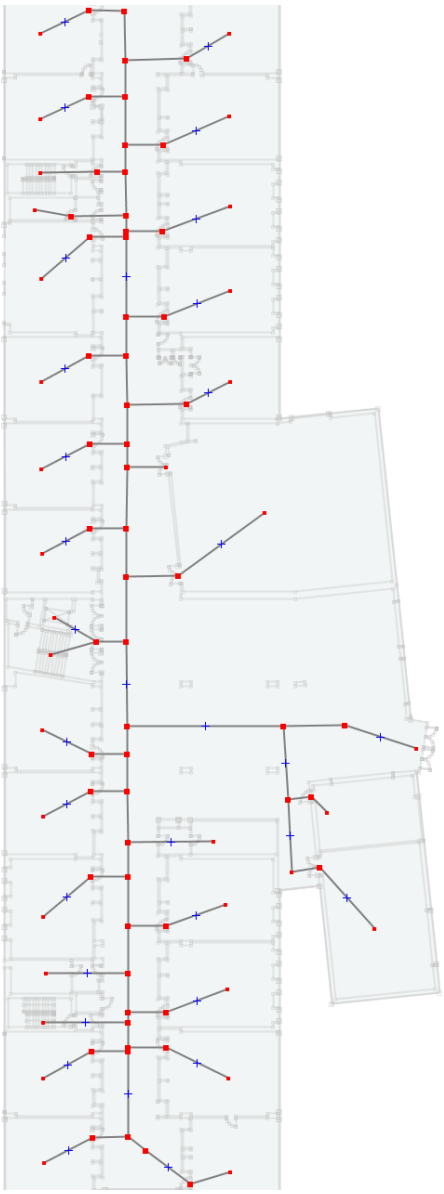
Koda 4.8 prikazuje kodo JavaScript, ki generira zemljevid Leaflet z določenim središčem (`center`), povečavo (`zoom`) in mejami vlečenja (`maxBounds`) ter tri plasti – vsako za eno nadstropje stavbe FRI. Zemljevid veže na element `<div>` z atributom `id='map'`.

Koda 4.8: Koda za generiranje načrta

```
var map = L.map('map', {
  center: [46.049334759334862, 14.46914668604788],
  zoom: 20,
  zoomControl: false,
  maxBounds: [[46.0489071152779,14.46688292154926],
              [46.04998108971369,14.4715286134519 25]],
});

var nad1 = L.tileLayer('tiles/tiles1/{z}/{x}/{y}.png', {
  maxZoom: 23,
  minZoom: 19,
});

var nad2 = L.tileLayer('tiles/tiles2/{z}/{x}/{y}.png', {
  maxZoom: 23,
  minZoom: 19,
});
var nad3 = L.tileLayer('tiles/tiles3/{z}/{x}/{y}.png', {
  maxZoom: 23,
  minZoom: 19,
});
```



Slika 4.4: Grafična ponazoritev grafa z vozlišči in povezavami v OSM.

Poglavje 5

Zasnova z uporabo tehnologij ANNE

V tem poglavju bomo predstavili zasnovo spletne aplikacije z uporabo skupine tehnologij za razvoj spletnih aplikacij ANNE na strani strežnika ter zasnovo podatkovne baze.

5.1 Implementacija grafa poti v Neo4j

Pri razvoju spletne aplikacije FRI Map smo v podatkovno bazo Neo4j shranili graf poti po stavbi Fakultete za računalništvo in informatiko.

Vozlišča grafa, implementiranega v podatkovni bazi Neo4j, predstavljajo lokacije v stavbi FRI. Priredimo jim oznako `Location`. Vsako vozlišče je identificirano z avtomatsko generiranim ID-jem ter vsebuje naslednje atribute:

- `x`: geografska širina lokacije na načrtu v decimalnih stopinjah.
- `y`: geografska dolžina lokacije na načrtu v decimalnih stopinjah.
- `Floor`: nadstropje, v katerem se lokacija nahaja (1, 2 ali 3).
- `Label`: atribut pove, ali lokacija predstavlja mesto na načrtu, na katerem bomo v aplikaciji prikazali oznako prostora (Yes ali No).

Vozlišča, ki predstavljajo vhode v prostore stavbe FRI, vsebujejo še naslednje atribute:

- `LocationName`: ime prostora v obliki R<št. nadstropja>.<št. prostora>, npr. R1.23.
- `Name`: dolgo ime prostora, npr. Predavalnica 1.
- `ShortName`: kratko ime prostora, npr. P01.
- `Type`: tip prostora (Predavalnica, Kabinet, Laboratorij, Ostalo).

Povezave predstavljajo pot med eno in drugo točko na načrtu. So tipa `CONNECTED_TO` in vsebujejo atribut `distance`, ki predstavlja razdaljo med točkama na načrtu. Ta razdalja je evklidska razdalja med točkama, izračunana po formuli (5.1).

$$distance\,(v1, v2) = \sqrt{(x\,(v1) - x\,(v2))^2 + (y\,(v1) - y\,(v2))^2} \tag{5.1}$$

Graf v podatkovni bazi Neo4j smo ustvarili tako, da smo iz datotek OSM z grafi poti po stavbi FRI v obliki OpenStreetMap in datoteke CSV, ki je vsebovala podatke o prostorih, generirali ustrezne stavke `CREATE` v poizvedovalnem jeziku Cypher. Del stavkov, ki ustvari dve vozlišči in povezavo med njima, je zapisanih v kodi 5.1

Koda 5.1: Stavki Cypher za generiranje vozlišč in povezav

```
CREATE (n20375: Location {
LocationName: 'R1.30', Floor: 1, Label: 'No',
x: 46.04936265787, y: 14.4699527841, Type: 'Predavalnica',
Name: 'Predavalnica 6', ShortName: 'P06'})

CREATE (n20373: Location {
LocationName: '', Floor: 1, Label: 'No',
x: 46.04943084589, y: 14.4699506186, Type: '',
Name: '', ShortName: ''})

CREATE (n20375)-[:CONNECTED_TO {distance:
sqrt((n20375.x - n20373.x )^2 + (n20375.y - n20373.y )^2)
}]->(n20373)
```

Slika 5.1 prikazuje del grafa poti z vsemi atributi vozlišča R1.30.

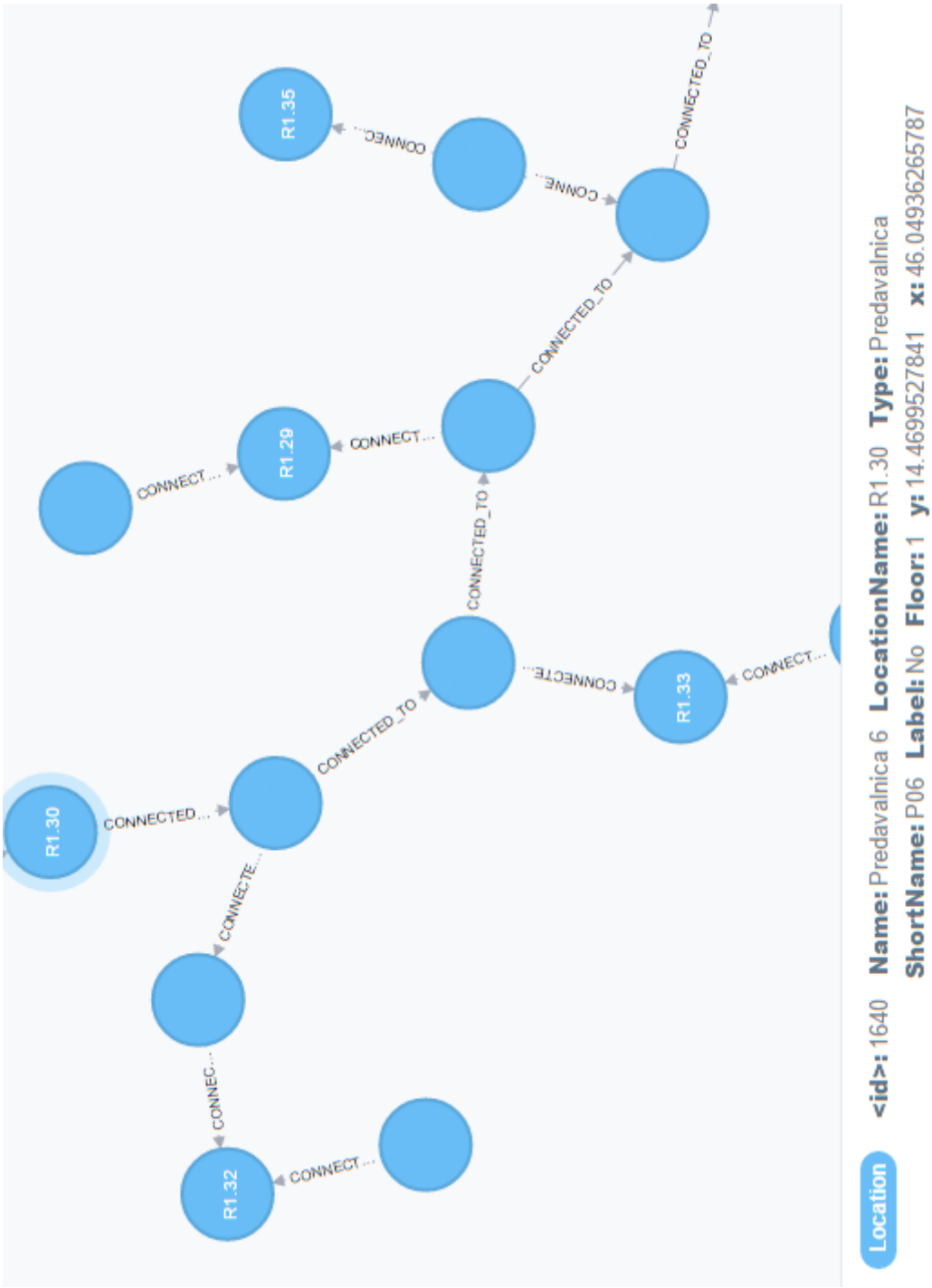
5.1.1 Iskanje najkrajše poti

Najkrajšo pot od točke x do točke y v grafu poti po stavbi Fakultete za računalništvo in informatiko smo dobili s pomočjo vgrajenega algoritma `shortestPath`. Poizvedba v jeziku Cypher, ki poišče najkrajšo pot od predavalnice 2 do referata, je prikazana v kodi 5.2

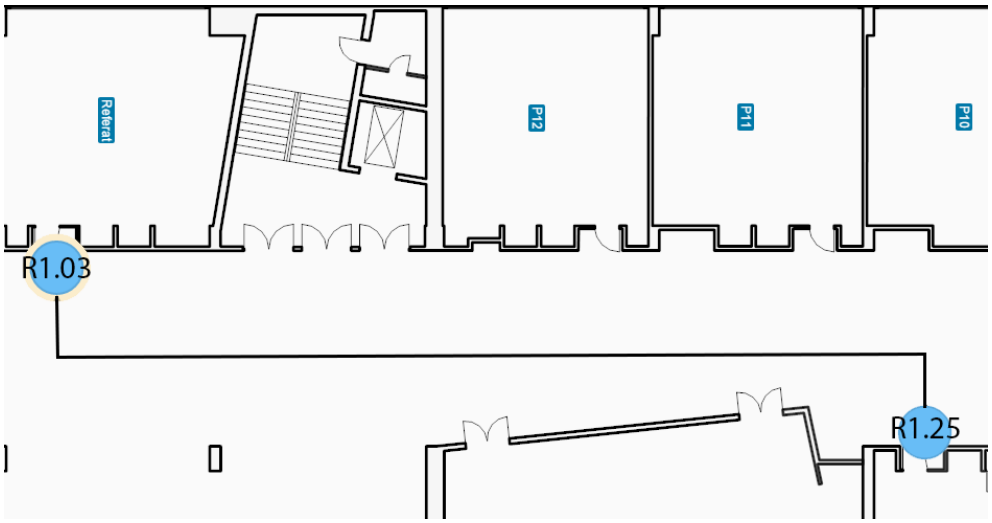
Koda 5.2: Poizvedba za iskanje najkrajše poti

```
MATCH (from:Location {LocationName:"R1.25"}),
(to:Location {LocationName:"R1.03"}),
path = shortestPath((from)-[:CONNECTED_TO*]-(to))
RETURN path
```

Da je iskanje v grafu potekalo v vse smeri, smo v poizvedbi ignorirali smeri povezav tipa `CONNECTED_TO`, kljub temu da ima v podatkovni bazi Neo4j vsaka povezava določeno smer. Z `*` smo povedali, da se lahko graf preiskuje v poljubno globino. Slika 5.2 prikazuje grafično ponazoritev poti, ki jo vrne poizvedba.



Slika 5.1: Del grafa poti. Prazna oznaka vozlišča pomeni segment hodnika.



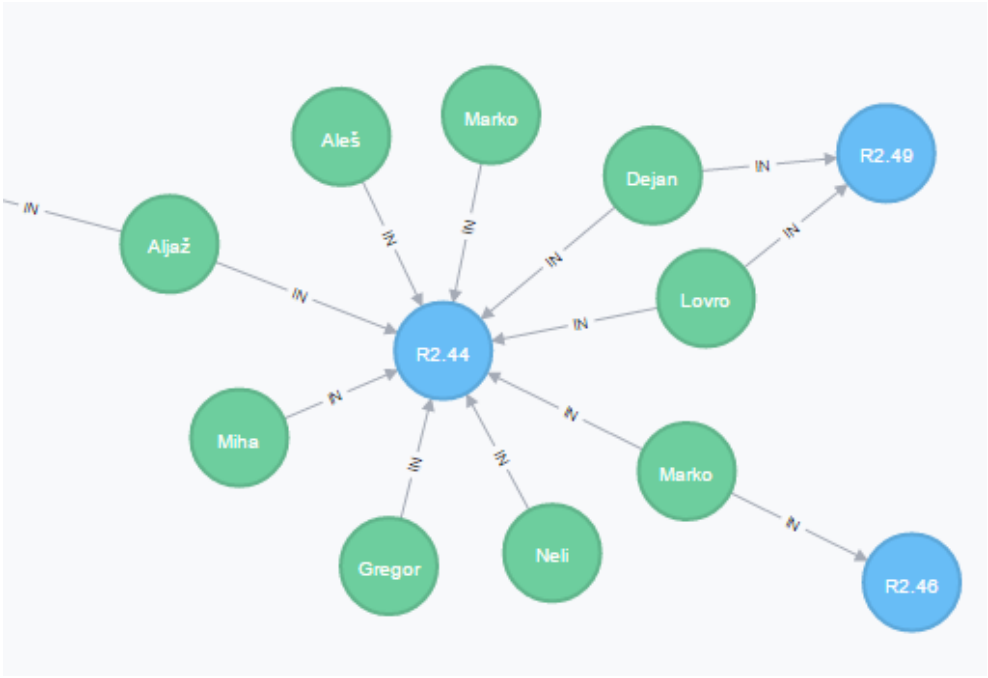
Slika 5.2: Grafična ponazoritev najkrajše poti.

5.2 Podatki o zaposlenih v Neo4j

Grafu v podatkovni bazi Neo4j smo dodali še podatke o zaposlenih na Fakulteti za računalništvo in informatiko. Vozliščem, ki predstavljajo zaposlene na FRI, priredimo oznako `Person`. Vsako vozlišče z oznako `Person` vsebuje avtomatsko generiran ID vozlišča in naslednje podatke:

- `FirstName`: ime zaposlenega.
- `LastName`: priimek zaposlenega.

Vsak zaposleni je s povezavo tipa `IN` povezan na vozlišče grafa z oznako `Location`, ki predstavlja vhod v prostor, kateremu zaposleni pripada. Grafična ponazoritev dela vozlišč, ki opisujejo pripadnost zaposlenih prostorom, je prikazana na sliki 5.3.



Slika 5.3: Grafična ponazoritev dela vozlišč, ki opisujejo pripadnost zaposlenih prostorom.

5.3 Aplikacija na strani strežnika

Celotna aplikacija na strani strežnika se nahaja v datoteki `server.js`. V aplikaciji na strani strežnika smo uporabili naslednje module Node.js:

- Express.js: ogrodje za vzpostavitev spletnega strežnika.
- node-neo4j: knjižnica za vzpostavitev povezave s podatkovno bazo Neo4j in pisanje poizvedb Cypher.
- body-parser [40]: knjižnica za pridobivanje teles zahtevkov HTTP.
- basic-auth-connect [41]: modul za avtentikacijo.

Koda 5.3 prikazuje, kako smo z uporabo ogrodja Express.js na vratih 3000 ustvarili spletni strežnik za uporabniško in administratorjevo dostopno točko,

ki je z uporabo modula `basic-auth-connect` zaščitena z uporabniškim imenom `admin` in geslom `admin`.

Koda 5.3: Spletni strežnik z uporabo Express.js.

```
var app = express();
app.use(express.static(__dirname + '/public'));
var auth = basicAuth('admin', 'admin');
app.use("/adm", [auth, express.static(__dirname + "/adm")]);
app.listen(3000);
```

V aplikaciji na strani strežnika smo se povezali na podatkovno bazo in obdelovali zahteve HTTP tipa `GET`, `POST`, `PUT` in `DELETE`.

Koda 5.4 prikazuje primer zahtevka `GET`, ki vrne pot od začetne do končne točke v formatu JSON. Zahtevek vsebuje dva parametra (x, y), ki predstavljata atributa `LocationName` začetne in končne točke.

Koda 5.4: Obdelava zahteve HTTP GET, ki vrne najkrajšo pot.

```
app.get('/path/:x/:y', function (req, res) {
  var x = req.params.x;
  var y = req.params.y;

  db.cypherQuery("MATCH (from: Location \
  {LocationName:'" + x +"'}), \
  (to: Location {LocationName:'" + y + "'}), \
  path = shortestPath((from)-[:CONNECTED_TO*]-(to)) \
  RETURN NODES(path)", function(err, result) {
    if(err) throw err;
    res.json(result.data);
  });
});
```

Koda 5.5 prikazuje primer zahtevka `DELETE`, ki iz podatkovne baze izbriše zaposlenega. ID zaposlenega posredujemo preko parametra.

Koda 5.5: Obdelava zahteve HTTP DELETE, ki izbriše zaposlenega.

```
app.delete('/adm/persons/:id', function (req, res) {
  var id = req.params.id;

  db.cypherQuery("MATCH (p:Person) WHERE Id(p) = " + id +
  + " DETACH DELETE p", function(err, result){
    if(err) throw err;
    res.json(result);
  });
});
```

Koda 5.6 prikazuje primer zahtevka `PUT`, ki posodobi ime in priimek zaposlenega. Podatki o zaposlenem (ID, novo ime, nov priimek) se prenesejo v telesu zahtevka HTTP.

Koda 5.6: Obdelava zahteve HTTP PUT, ki posodobi zaposlenega.

```
app.put('/adm/persons/:id', function (req, res) {
  db.cypherQuery("MATCH (p:Person) WHERE Id(p) = " +
  + req.params.id  + " SET p.FirstName = '" +
  + req.body.FirstName + "', p.LastName = '" +
  + req.body.LastName + "'", function(err, result) {
    if(err) throw err;
    res.json(result);
  });
});
```

Koda 5.7 prikazuje primer zahtevka `POST`, ki v podatkovno bazo doda novega zaposlenega. Podatki o zaposlenem (ime, priimek) se prenašajo v telesu zahtevka HTTP.

Koda 5.7: Obdelava zahteve HTTP POST, ki doda zaposlenega.

```
app.post('/adm/persons', function (req, res) {
  db.cypherQuery("CREATE ( p:Person { FirstName: '" +
  + req.body.FirstName + "', LastName: '" +
  + req.body.LastName + "' } )", function(err, result) {
    if(err) throw err;
    res.json(result.data);
  });
});
```

Poglavje 6

Zasnova z uporabo tehnologij MEAN

V tem poglavju bomo predstavili zasnovo spletne aplikacije z uporabo skupine tehnologij za razvoj spletnih aplikacij MEAN na strani strežnika ter zasnovo podatkovne baze.

6.1 Implementacija grafa poti v MongoDB

Pri razvoju spletne aplikacije FRI Map smo v podatkovno bazo MongoDB v primerni notaciji za nadaljnjo obdelavo shranili graf poti po stavbi Fakultete za računalništvo in informatiko. Celoten graf smo zapisali v zbirko z imenom `graph` v en dokument v obliki, ki implementira redko matriko in je prikazana v kodi 6.1.

Koda 6.1: Graf v MongoDB

```
{'vozlisce1': {'vozlisce2': razdalja do vozlisca 2},
'vozlisce2': {'vozlisce1': razdalja do vozlisca 1,
              'vozlisce3': razdalja do vozlisca 3},
'vozlisce3': { ... },
...}
```

Graf je torej zapisan kot seznam vseh vozlišč, za vsako vozlišče pa so navedena povezana vozlišča z razdaljami, ki so izračunane po evklidski razdalji.

Za vsako vozlišče grafa smo nato v zbirki `graph` ustvarili dokument s podatki o vozlišču:

- `_id`: oznaka lokacije, ki se ujema z oznako vozlišča v grafu.
- `x`: geografska širina lokacije na načrtu v decimalnih stopinjah.
- `y`: geografska dolžina lokacije na načrtu v decimalnih stopinjah.
- `Floor`: nadstropje, v katerem se lokacija nahaja (1, 2 ali 3).
- `Label`: atribut pove, ali lokacija predstavlja mesto na načrtu, na katerem bomo v aplikaciji prikazali oznako prostora (Yes ali No).

Dokumenti, ki predstavljajo lokacije, ki so vhodi v prostore stavbe FRI, vsebujejo še naslednje atribute:

- `LocationName`: ime prostora v obliki R<št. nadstropja>.<št. prostora>, npr. R1.23.
- `Name`: dolgo ime prostora, npr. Predavalnica 1.
- `ShortName`: kratko ime prostora, npr. P01.
- `Type`: tip prostora (Predavalnica, Kabinet, Laboratorij, Ostalo).

Slika 6.1 prikazuje del podatkov v zbirki `graph`. V dokumentu z `_id = 1` se nahaja graf poti, v ostalih dokumentih pa podatki o vozliščih grafa.

6.1.1 Iskanje najkrajše poti

Iskanje najkrajše poti od točke x do točke y v grafu poti po stavbi Fakultete za računalništvo in informatiko smo rešili programsko na strani strežnika.

```
{ "_id" : 1, "graph" : "{ 'n20209': { 'n20301':  0.0000845357, },
'n20207': { 'n20289':  0.0000490047, },
'n20205': { 'n20263':  0.0000300150, },
'n20203': { 'n20265':  0.0001151968, },
'n20201': { 'n20329':  0.0000552738, 'n20673': 0.0000000001, },
'n20199': { 'n20329':  0.0000608431, 'n20889': 0.0000000001, },
'n20197': { 'n20281':  0.0000599175, 'n20283':  0.0000421596, },
'n20899': { 'n20611':  0.0000376133, 'n20609':  0.0000256222, } ... }"}

{ "_id" : "n20377", "LocationName" : "", "Name" : "", "Floor" : 1,
"y" : 14.46922810091, "x" : 46.0492759053, "ShortName" : "", "Type" : "",
"Label" : "Yes" }

{ "_id" : "n20375", "LocationName" : "R1.30", "Name" : "Predavalnica 6",
"Floor" : 1, "y" : 14.4699527841, "x" : 46.04936265787, "ShortName" : "P06",
"Type" : "Predavalnica", "Label" : "No" }

{ "_id" : "n20373", "LocationName" : "", "Name" : "", "Floor" : 1,
"y" : 14.4699506186, "x" : 46.04943084589, "ShortName" : "", "Type" : "",
"Label" : "No" }

{ "_id" : "n20371", "LocationName" : "", "Name" : "", "Floor" : 1,
"y" : 14.47002853946, "x" : 46.04943094217, "ShortName" : "", "Type" : "",
"Label" : "No" }
```


Slika 6.1: Del podatkov zbirke graph.

Izvedli smo poizvedbo, s katero smo dobili shranjeni graf, ki vsebuje oznake vozlišč in razdalje. Na strežniku smo nad grafom nato izvedli algoritem Dijkstra, ki je vrnil zaporedje oznak vozlišč od točke x do točke y. Nazadnje smo izvedli še poizvedbe, ki so vsebovale oznake teh vozlišč, da smo iz dokumentov baze dobili podatke o vozliščih.

6.2 Podatki o zaposlenih v MongoDB

Poleg podatkov o vozliščih grafa smo v podatkovno bazo MongoDB shranili še podatke o zaposlenih na Fakulteti za računalništvo in informatiko. Zaposlene smo shranili v zbirko `person`. Vsak dokument zbirke `person` vsebuje naslednje atribute:

- `_id`: avtomatsko generiran ID zaposlenega.
- `FirstName`: ime zaposlenega.

- `LastName`: priimek zaposlenega.
- `Locations`: seznam ID-jev lokacij, katerim pripada zaposleni.

6.3 Aplikacija na strani strežnika

Zasnova aplikacije na strani strežnika z uporabo skupine tehnologij MEAN je podobna kot pri tehnologijah ANNE. Namesto knjižnice node-neo4j smo za povezavo s podatkovno bazo MongoDB uporabili knjižnico mongojs [24]. Uporabili smo še knjižnico node-dijkstra [42], s katero smo nad grafom poti, shranjenim v podatkovni bazi MongoDB, izvedli Dijkstrov algoritem, ki nam je vrnil zaporedje oznak vozlišč najkrajše poti med lokacijama v stavbi.

Obravnavali smo iste zahtevke HTTP kot pri aplikaciji, narejeni s skupino tehnologij ANNE.

Koda 6.2 prikazuje obravnavo zahtevka `GET`, ki vrne najkrajšo pot od začetne do končne točke. Preko parametrov posredujemo atributa `LocationName` začetne in končne točke (spremenljivki x, y). Nato izvedemo poizvedbi, da pridobimo `_id` teh dveh vozlišč (spremenljivki a, b). Iz podatkovne baze preberemo graf, ki se nahaja pod `_id = 1`. Nad grafom izvedemo Dijkstrov algoritem s knjižnico node-dijkstra. Knjižnica iz objekta JavaScript tipa `Object`, ki vsebuje graf v obliki redke matrike, zgradi objekt tipa `Graph`. Funkcija `path()` nam v zaporedju vrne ID-je vozlišč, ki sestavljajo najkrajšo pot od začetne do končne točke (spremenljivka path). Nato izvedemo poizvedbo, ki nam vrne dokumente, ki predstavljajo ta vozlišča. Uporabimo operator `$in`, s katerim lahko v poizvedbi uporabimo seznam ID-jev vozlišč, ki sestavljajo najkrajšo pot. Ker MongoDB ob uporabi operatorja `$in` ne zagotavlja vračila podatkov v istem vrstnem redu, kot so bili ID-ji v seznamu, podatke še uredimo v pravilen vrstni red, kakršen je v spremenljivki `path`, in rezultat vrnemo v formatu JSON.

Koda 6.2: Obdelava zahteve HTTP GET, ki vrne najkrajšo pot.

```
app.get('/path/:x/:y', function (req, res) {
  var x = req.params.x;
  var y = req.params.y;
  var a = null;
  var b = null;
// poizvedba, s katero dobimo ID tocke x in ga shranimo v a
  db.graph.findOne({LocationName: x}, function(err, docs) {
    a = docs._id;
// poizvedba, s katero dobimo ID tocke y in ga shranimo v b
    db.graph.findOne({LocationName: y}, function(err, docs) {
      b = docs._id;
// poizvedba, ki vrne graf; shranimo ga v spremenljivko graf
      db.graph.findOne({
        _id: 1
      }, function(err, docs) {
        var graf = docs.graph;
// vrednost tipa String v graf zapisemo kot Object v graf2
        eval('var graf2='+graf);
// Dijkstrov algoritem z modulom node-dijkstra
        const Graph = require('node-dijkstra');
        const route = new Graph(graf2);
        var path = route.path(a, b);
// poizvedba, ki vrne seznam lokacij v poti
        db.graph.find({
          _id: { $in: path }
        }, function(err, docs) {
// uredimo po vrstnem redu, kot je v spremenljivki path
          var hash = createHashOfResults(docs);
          var results = [];
          for(var i = 0 ; i < path.length ; i++){
            var queryId = path[i];
            var result = hash[queryId];
            results.push(result);
          }
          res.json(results);
...
```

Koda 6.3 prikazuje primer zahtevka `DELETE`, ki iz podatkovne baze izbriše zaposlenega. ID zaposlenega posredujemo preko parametra.

Koda 6.3: Obdelava zahteve HTTP DELETE, ki izbriše zaposlenega.

```
app.delete('/adm/persons/:id', function (req, res) {
  var id = req.params.id;
  db.person.remove({
    _id: mongojs.ObjectId(id)
  }, function(err, docs) {
    res.json(docs);
  });
});
```

Koda 6.4 prikazuje primer zahtevka `PUT`, ki posodobi ime in priimek zaposlenega. Podatki o zaposlenem (ID, novo ime, nov priimek) se prenesejo v telesu zahtevka HTTP.

Koda 6.4: Obdelava zahteve HTTP PUT, ki posodobi zaposlenega.

```
app.put('/adm/persons/:id', function (req, res) {
  var id = req.params.id;
  db.person.update({
    _id: mongojs.ObjectId(id)
  },{
    $set: {
      FirstName: req.body.FirstName,
      LastName: req.body.LastName
    }
  }, function(err, docs) {
    res.json(docs);
  });
});
```

Koda 6.5 prikazuje primer zahtevka `POST`, ki v podatkovno bazo doda novega zaposlenega. Podatki o zaposlenem (ime, priimek) se prenašajo v telesu zahtevka HTTP.

Koda 6.5: Obdelava zahteve HTTP POST, ki doda zaposlenega.

```
app.post('/adm/persons', function (req, res) {
  db.person.insert({
    FirstName: req.body.FirstName,
    LastName: req.body.LastName,
    Locations: []
  }, function(err, docs) {
    res.json(docs);
  });
});
```

Poglavje 7

Zasnova z uporabo tehnologij LAMP

V tem poglavju bomo opisali, kako bi spletno aplikacijo zasnovali z uporabo tehnologij LAMP.

7.1 Implementacija grafa poti v MySQL

Graf poti bi morali implementirati v podatkovni bazi MySQL. Ena možnih implementacij bi bila sledeča: v eno tabelo bi shranili vsa vozlišča grafa s podatki o lokacijah, v drugo pa seznam vseh povezav med vozlišči z razdaljami. Za iskanje najkrajše poti bi uporabili shranjeno proceduro SQL. Ustvarili bi še tabelo s podatki o zaposlenih. Tabela z zaposlenimi in tabela z vozlišči grafa bi bili povezani z relacijo m:n, ki bi predstavljala pripadnost zaposlenih prostorom.

7.2 Aplikacija na strani strežnika

Strežnik bi odjemalcu vračal podatke iz podatkovne baze v formatu JSON. V programskem jeziku PHP bi napisali REST API, ki bi se povezal na podatkovno bazo MySQL in obdeloval zahteve HTTP. Ob zahtevi bi rezultate

iz baze podatkov MySQL vračali v formatu JSON v popolnoma enaki obliki kot pri aplikaciji, narejeni s skupinama tehnologij ANNE ali MEAN.

7.3 Aplikacija na strani odjemalca

Na strani odjemalca bi uporabili iste knjižnice JavaScript kot pri zasnovi s skupinama tehnologij ANNE ali MEAN. Aplikacija na strani odjemalca bi od strežnika prejela seznam prostorov in zaposlenih v formatu JSON. Ob zahtevi za iskanje poti bi strežnik klical shranjeno proceduro SQL, ki bi vrnila najkrajšo pot, in jo posredoval odjemalcu v obliki JSON. Ker bi zahteve HTTP vračale iste rezultate, kot pri aplikaciji, narejeni s skupinama tehnologij ANNE ali MEAN, bi to pomenilo, da bi bila aplikacija na strani odjemalca popolnoma enaka.

Poglavje 8

Zaključek

Iz praktičnega primera je razvidno, da lahko z uporabo vsake od opisanih skupin tehnologij brez težav razvijemo sodobno spletno aplikacijo. Skupine tehnologij za razvoj spletnih aplikacij, predstavljene v tem diplomskem delu, pa vsekakor niso edine, ki jih lahko uporabimo za razvoj spletne aplikacije.

Pojavlja se vprašanje, ali so skupine tehnologij za razvoj spletnih aplikacij sploh potrebne. Obstaja mnogo alternativ – za uporabo programskega jezika JavaScript na strani strežnika bi lahko namesto Node.js uporabili izvajalno okolje JXcore [43]. S strežnikom Apache bi lahko namesto PHP-ja uporabili Python ali Perl. Tudi pri podatkovnih bazah imamo na voljo mnogo alternativ, tako relacijskih kot nerelacijskih. Še največ možnosti pa imamo na strani odjemalca, kjer nam je poleg AngularJS na voljo še mnogo drugih knjižnic JavaScript, med katerimi so najpopularnejše Meteor, React, jQuery in Fundation [44].

Pridemo lahko do naslednjega zaključka: skupino tehnologij za razvoj spletnih aplikacij si sestavimo sami glede na potrebe naše spletne aplikacije in glede na podatke, ki jih aplikacija hrani. Na praktičnem primeru spletne aplikacije FRI Map smo prikazali, da je bila implementacija grafa poti po notranjosti stavbe in iskanje po grafu najenostavnejše z grafno podatkovno bazo Neo4j, saj smo lahko graf poti v podatkovno bazo shranili kar v osnovni obliki. Uporabili pa bi lahko tudi katerokoli drugo podatkovno bazo, za

katero obstaja knjižnica za povezavo.

Omejitve se pojavijo, če za postavitev spletne aplikacije uporabljamo ponudnika gostovanja. Večina ponudnikov gostovanja v svojih osnovnih paketih ponuja skupino tehnologih LAMP, nekaj pa jih ponuja tudi skupino tehnologij MEAN. Za uporabo drugih poljubnih kombinacij tehnologij moramo izbrati dražje gostovanje VPS (Virtual Private Server — navidezni zasebni strežnik), s katerim imamo popoln dostop do strežnika, s katerim lahko namestimo poljubno programsko opremo.

Razvoj spletne aplikacije z modernejšima skupinama tehnologij ANNE in MEAN je v primerjavi s skupino LAMP lažji in manj zamuden. Na strani strežnika je bilo z uporabo ogrodja Express.js mogoče napisati funkcije za obdelavo zahtev HTTP v le nekaj vrsticah, obdelava zahtev z uporabo skupine LAMP zahteva več kode in dodatne nastavitve na spletnem strežniku. Tudi na strani odjemalca nam AngularJS s pristopom model-pogled-kontrolnik omogoča, da imamo vse tri komponente ločene. To in dejstvo, da sta podatkovni bazi Neo4j in MongoDB veliko bolj fleksibilni kot relacijske podatkovne baze, pripomore k temu, da je kakršnekoli spremembe v spletni aplikaciji veliko lažje implementirati. Node.js zaradi svoje dogodkovne arhitekture hitreje obdeluje večje število sočasnih zahtev kot Apache [45]. Nerelacijske podatkovne baze določene poizvedbe obdelujejo hitreje kot relacijske, kar pomeni, da lahko spletne aplikacije, narejene z uporabo tehnologij ANNE in MEAN, tečejo hitreje kot aplikacije, narejene s tehnologijami LAMP. Prednost skupin ANNE in MEAN je tudi uporaba programskega jezika JavaScript na obeh straneh razvoja.

V diplomskem delu seveda nismo raziskali vseh možnosti uporabe skupin tehnologij ANNE in MEAN. Skupina tehnologij LAMP obstaja bistveno dlje in za tehnologije LAMP obstaja bistveno večje število bolje razvitih ogrodij in modulov.

Na podlagi pridobljenih izkušenj bi priporočal uporabo modernejših skupin tehnologij ANNE in MEAN, menim pa, da bi moralo biti glavno vodilo pri izbiri skupine tehnologij za razvoj spletnih aplikacij to, kakšne so zahteve

spletne aplikacije in kakšne podatke bomo hranili v podatkovni bazi.

Literatura

[1] LAMP Stack as new program development paradigm. [Online]. Dosegljivo: `http://www.softpanorama.org/WWW/lamp.shtml`. [Dostopano 15. 11. 2015].

[2] G. Lawton. "LAMP lights enterprise development efforts", *IEEE Computer*, št. 38, zv. 9, str. 18–20, 2005.

[3] C. J. Ihrig, A. Bretz. *Full Stack JavaScript Development With MEAN, 1st Edition*. Kevin, 2014.

[4] JavaScript. [Online]. Dosegljivo: `https://en.wikipedia.org/wiki/JavaScript`. [Dostopano 15. 11. 2015].

[5] ECMA-262, Edition 5. [Online]. Dosegljivo: `http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf`. [Dostopano 29. 1. 2016].

[6] Chrome V8. [Online]. Dosegljivo: `http://https://developers.google.com/v8/`. [Dostopano 29. 1. 2016].

[7] Node.js. [Online]. Dosegljivo: `https://en.wikipedia.org/wiki/Node.js`. [Dostopano 15. 11. 2015].

[8] Making Node.JS Work With Apache. [Online]. Dosegljivo: http://blog.podrezo.com/making-node-js-work-with-apache/. [Dostopano 29. 1. 2016].

[9] npmjs. [Online]. Dosegljivo: https://www.npmjs.com/. [Dostopano 29. 1. 2016].

[10] Ajax. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Ajax_(programming). [Dostopano 29. 11. 2015].

[11] Single-page application. Dosegljivo: https://en.wikipedia.org/wiki/Single-page_application. [Dostopano 29. 11. 2015].

[12] A. Q. Haviv. *MEAN Web Development*. Packt Publishing Ltd, 2014.

[13] ANNE Stack – Angular JS, Node, Neo4J and Express. [Online]. Dosegljivo: http://www.42id.com/articles/anne-stack-angular-js-node-neo4j-and-express/. [Dostopano 29. 11. 2015].

[14] AngularJS. [Online]. Dosegljivo: https://angularjs.org/. [Dostopano 20. 11. 2015].

[15] Model-View-Controller. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Model-view-controller. [Dostopano 29. 1. 2016].

[16] Linux. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Linux. [Dostopano 26. 1. 2016].

[17] Usage statistics and market share of Linux for websites. [Online]. Dosegljivo: http://w3techs.com/technologies/details/os-linux/all/all. [Dostopano 26. 1. 2016].

[18] Why Linux Beats Windows for Servers. [Online]. Dosegljivo: `http://www.pcworld.com/article/204423/why_linux_beats_windows_for_servers.html`. [Dostopano 26. 1. 2016].

[19] Apache HTTP Server. [Online]. Dosegljivo: `https://en.wikipedia.org/wiki/Apache_HTTP_Server`. [Dostopano 18. 1. 2016].

[20] MySQL Reference Manual. [Online]. Dosegljivo: `http://dev.mysql.com/doc/refman/5.7/en/`. [Dostopano 26. 1. 2016].

[21] PHP Manual. [Online]. Dosegljivo: `https://secure.php.net/manual/en/`. [Dostopano 18. 1. 2016].

[22] S. Holzner. *PHP: The Complete Reference*. McGraw Hill Professional, 2008.

[23] node-neo4j. [Online]. Dosegljivo: `https://github.com/philippkueng/node-neo4j`. [Dostopano 15. 11. 2015].

[24] mongojs. [Online]. Dosegljivo: `https://www.npmjs.com/package/mongojs`. [Dostopano 15. 11. 2015].

[25] AngularJS Developer Guide. [Online]. Dosegljivo: `https://docs.angularjs.org/guide`. [Dostopano 21. 11. 2015].

[26] Neo4j. [Online]. Dosegljivo: `http://neo4j.com/`. [Dostopano 20. 11. 2015].

[27] Neo4j Developer Guide. [Online]. Dosegljivo: `http://neo4j.com/developer/get-started/`. [Dostopano 15. 1. 2016].

[28] What is a Graph Database? [Online]. Dosegljivo: http://neo4j.com/developer/graph-database/. [Dostopano 15. 1. 2016].

[29] S. Batra, C. Tyagi. "Comparative Analysis of Relational And Graph Databases", *International Journal of Soft Computing and Engineering*, št. 2, zv. 2, 2012.

[30] Neo4j Manual. [Online]. Dosegljivo: http://neo4j.com/docs/stable/. [Dostopano 21. 11. 2015].

[31] Node.js. [Online]. Dosegljivo: https://nodejs.org/en/. [Dostopano 20. 11. 2015].

[32] Express.js. [Online]. Dosegljivo: http://expressjs.com/. [Dostopano 20. 11. 2015].

[33] Express.js Guide. [Online]. Dosegljivo: http://expressjs.com/en/guide. [Dostopano 21. 11. 2015].

[34] MongoDB. [Online]. Dosegljivo: https://www.mongodb.org/. [Dostopano 20. 11. 2015].

[35] DB-Engines Ranking. [Online]. Dosegljivo: http://db-engines.com/en/ranking. [Dostopano 29. 1. 2016].

[36] MongoDB Manual. [Online]. Dosegljivo: https://docs.mongodb.org/manual. [Dostopano 12. 12. 2015].

[37] R. Emeršič. *Mobilna aplikacija za prikaz prostorov in načrtovanje poti v stavbah*. FRI, 2015.

[38] Maperative. [Online]. Dosegljivo: http://maperitive.net/. [Dostopano 15. 11. 2015].

[39] Leaflet. [Online]. Dosegljivo: http://leafletjs.com/. [Dostopano 15. 11. 2015].

[40] body-parser. [Online]. Dosegljivo: `https://github.com/expressjs/body-parser`. [Dostopano 15. 11. 2015].

[41] basic-auth-connect. [Online]. Dosegljivo: `https://www.npmjs.com/package/basic-auth-connect`. [Dostopano 15. 11. 2015].

[42] node-dijkstra. [Online]. Dosegljivo: `https://www.npmjs.com/package/node-dijkstra`. [Dostopano 15. 11. 2015].

[43] JXCore. [Online]. Dosegljivo: `http://jxcore.com/`. [Dostopano 6. 2. 2016].

[44] Javascripting Popular. [Online]. Dosegljivo: `https://www.javascripting.com/`. [Dostopano 6. 2. 2016].

[45] Benchmarking Node.js - basic performance tests against Apache + PHP. [Online]. Dosegljivo: `http://zgadzaj.com/benchmarking-nodejs-basic-performance-tests-against-apache-php`. [Dostopano 13. 2. 2016].

Dodatki

Dodatek A

Navodila za namestitev

V tem dodatku sledijo navodila za namestitev spletne aplikacije FRI Map, narejene s skupinama tehnologij ANNE in MEAN, na operacijski sistem Windows.

1. S spletne strani `https://nodejs.org/en/download/` prenesemo inštalacijsko datoteko Node.js Stable za Windows.
   S spletne strani `https://www.mongodb.org/downloads` prenesemo inštalacijsko datoteko MongoDB za Windows.
   S spletne strani `http://neo4j.com/download/` prenesemo inštalacijsko datoteko Neo4j Community Edition za Windows.

2. Zaženemo inštalacijsko datoteko Node.js in sledimo čarovniku za namestitev.

3. Zaženemo inštalacijsko datoteko MongoDB. MongoDB namestimo na mesto `C:\mongodb`.

4. Zaženemo inštalacijsko datoteko Neo4j in sledimo čarovniku za namestitev.

5. Mapo `\db` s podatki podatkovne baze MongoDB shranimo na mesto `C:\data`.

6. Mapo `\default.graphdb` s podatki podatkovne baze Neo4j shranimo na mesto `C:\data`.

7. Mapi `\frimap` in `\frimap2` shranimo na mesto `C:\nodeprojects`.

8. Odpremo ukazno vrstico in poženemo `C:\mongodb\bin\mongod.exe`. Okno ukazne vrstice pustimo odprto.

9. Zaženemo aplikacijo Neo4j Community Edition in izberemo mapo `C:\data\default.graphdb`. Kliknemo na gumb Start.

10. Za zagon spletne aplikacije, narejene s skupino tehnologij ANNE, odpremo novo okno ukazne vrstice in se pomaknemo v mapo `C:\nodeprojects\frimap`. Vpišemo ukaz `node server.js`. Izpiše se 'Server running on 3000'. Spletna aplikacija je dostopna na naslovu `http://localhost:3000`.

11. Za zagon spletne aplikacije, narejene s skupino tehnologij MEAN, odpremo novo okno ukazne vrstice in se pomaknemo v mapo `C:\nodeprojects\frimap2`. Vpišemo ukaz `node server.js`. Izpiše se 'Server running on 3000'. Spletna aplikacija je dostopna na naslovu `http://localhost:3000`.